CONTROL DATA
CORPORATION

CONTROL DATA®
STAR COMPUTER SYSTEM

STAR ASSEMBLER REFERENCE MANUAL

CONTROL DATA CORPORATION

TITLE: STAR ASSEMBLER Reference Manual

PUBLICATION NO. 19980200
REVISION B

DATE: November 20, 1974

REASON FOR CHANGE:

Revised to clarify and expand the explanation of certain topics. Corrections and comments noted by readers have been incorporated. Examples in Appendix I have been replaced by those reflecting use of Assembler Version 2.2.

INSTRUCTIONS:

This revision constitutes a complete reprint and obsoletes previous printings.

New features, as well as changes, deletions, and additions to information in this manual are indicated by bars in the margins or by a dot near the page number if the entire page is affected. A bar by the page number indicates pagination rather than content has changed.

REVISION RECORD

REVISION	DESCRIPTION
01	Original release
(7-27-73)	
A	Revised to incorporate an appendix pertaining to STAR OS and to update information contained in
(4-26-74)	this document.
B	Revised to clarify and expand the explanation of certain topics. Corrections and comments noted by
(12-1-74)	readers have been incorporated. Examples in appendix I have been replaced by those reflecting use of
	Assembler Version 2.2.
Publication No. 19980200	

Additional copies of this manual may be obtained from the nearest Control Data Corporation sales office.

Address comments concerning this manual to:

CONTROL DATA CORPORATION
Software Documentation
215 MOFFETT PARK DRIVE
SUNNYVALE, CALIFORNIA 94086

or use Comment Sheet in the back of this manual

PREFACE

This reference document discusses the principles, features, methods, rules and techniques of producing a CONTROL DATA® STAR Assembler Language program.

The reader is encouraged to study the subject matter in the order presented:

Section 1	Introduction – Introduces features of the STAR Assembler considered most important.
Section 2	Program Structure – Discusses the structure of a typical assembler program and introduces the assembler coding conventions.
Section 3	Statement Structure – Describes all assembler statement organization and rules.
Section 4	Directives – Details all available assembler directives and the organization of assembler procedures and functions. A directive summary is also provided.
Section 5	Assembler Provided Functions and Procedures – Details all functions and procedures provided as part of the STAR Assembler.
Appendix A	Elementary Items – Describes the data types permitted for use with the assembly language.
Appendix B	Expression – Describes the types of expressions permitted for use with the assembly language.
Appendix C	STAR Machine Instructions – Provides a more than cursory discussion of the machine instruction types and includes a summary list of all machine instructions with format and function descriptions.
Appendix D	JOB Processing Deck Structure
Appendix E	Assembly Listing Format – Describes and illustrates the format of an assembly listing.
Appendix F	Error Messages – Lists all error messages produced by the assembler.
Appendix G	Predefined Symbols – Lists all predefined assembler symbols, their values, and use.

Appendix H	Assembly Limitations – Lists assembler limitations.
Appendix I	Examples – Sample program descriptions.

Information supporting this document is given in the following publications:

STAR-100 Hardware Reference Manual Pub. No. 60256000

STAR-65 Hardware Reference Manual Pub. No. 19980000

STAR Computer System Operating System Reference Manual Pub. No. 60384400

CONTENTS

1 INTRODUCTION	1-1
Features	1-1
Procedures	1-2
Functions	1-2
Sets/Symbols	1-2
Attributes	1-3
Basic Program Structure	1-3
Assembly Process	1-4
Operating System	1-4
Assembler Error Detection	1-5
2 PROGRAM STRUCTURE	2-1
Assembler Code Conventions	2-2
Program Universal Area	2-2
Subprogram Area	2-3
Code Section	2-3
Data Section	2-3
Common Section	2-3
Levels of Symbol Definition	2-4
Levels of Symbol Reference	2-4
3 STATEMENT STRUCTURE	3-1
4 DIRECTIVES	4-1
General	4-1
Input/Output Control	4-1
Input	4-2
Output	4-2
Listing	4-3
LIBP	4-3
Listing Control	4-4
SPACING	4-4
EJECT	4-4
TITLE	4-5
MESSAGE	4-5
NOLIST	4-5
LIST	4-5
DETAIL	4-6
BRIEF	4-6
Assembly Control	4-7
IDENT	4-7
END	4-7
FINIS	4-7
Conditional Assembly	4-7
RPT	4-8
GOTO	4-10
RPT and GOTO Processing	4-12
Subprogram Linking	4-13
ENTRY	4-13
Externals (EXTD, EXTC)	4-14
Symbol and Set Definition, and Referencing	4-16
SET	4-16
Referencing Sets	4-19
Element and Sub-Element Referencing	4-19
Assignment	4-20
RDEF	4-20
EQU	4-21
Data Generation	4-23
FORM	4-23
Form Referencing	4-24
GEN	4-25
Address and Location Control	4-27
Default MSEC	4-28
MSEC	4-29
RES	4-31
ORG	4-31
EORG	4-32
Attribute Control	4-33
RATT	4-33
Referencing Attributes	4-34
Procedures	4-34
Writing a Procedure	4-34
PROC	4-36
NAME (Procedure)	4-36
ENDP (Procedure)	4-37
Procedure Reference	4-38
Procedure Reference Termination, EXITP	4-38
Procedure Reference Function Flow	4-39
Functions	4-44
Function Definition	4-44
FUNC	4-45
NAME (Function)	4-45
Function References	4-46

ENDP (Function) 4-46
EXITP (Function) 4-46
Summary of Directives 4-50

5 ASSEMBLER PROVIDED FUNCTIONS AND PROCEDURES 5-1
Conversion Functions 5-1
Symbol Creation Function 5-3
Attribute Function 5-4
Intrinsic Attributes 5-4
ATT 5-5

GLOSSARY Glossary-1

APPENDIXES

A ELEMENTARY ITEMS A-1

B EXPRESSIONS B-1

C STAR MACHINE INSTRUCTIONS C-1

D JOB PROCESSING AND DECK STRUCTURE D-1

E ASSEMBLY LISTING FORMAT E-1

F ERROR MESSAGES F-1

G ASSEMBLER PREDEFINED COMMAND–SYMBOLS G-1

H ASSEMBLER LIMITATIONS H-1

I EXAMPLES I-1

FIGURES

2-1 Program Structure 2-1
4-1 Association of Procedure Definition and Reference Elements 4-43
4-2 Association of Function Definition and Reference Elements 4-49
B-1 Expression Hierarchial Evaluation B-9

TABLES

2-1 Symbol Levels 2-4
3-1 Statement Format 3-2
4-1 Summary of Directives 4-51
4-2 STAR Assembler Directive Parameters 4-55
5-1 Conversion Functions 5-2
A-1 STAR Character Set A-2
A-2 Delimiter Characters A-3
A-3 Special Characters A-4
A-4 Summary of Rules for Constants A-14
A-5 Symbol Summary A-17
B-1 Operators B-2
B-2 Comparison Methods B-3
B-3 Unary + – Operations B-4
B-4 Binary Scale Operations (.BS.) B-5
B-5 Multiply and Divide Operations (* /) B-5
B-6 Add and Subtract Operations (+ –) B-6
B-7 Relational Operations (EQ, NE, GT, GE, LT, LE) B-7
B-8 Logical Operations (AND, OR) B-8
C-1 Qualifiers C-2
C-2 Instruction Designators C-6
C-3 Vector Instruction Sub-function Bits C-11
C-4 Vector Instruction Sign Control Sub-function Bits C-12
C-5 String Instruction G Designators C-16
C-6 Instructions with Sign Control C-18
C-7 Index Instructions C-19
C-8 Register Instructions C-20
C-9 Branch Instructions C-23
C-10 Vector Instructions C-25

C-11	Sparse Vector Instructions	C-27
C-12	Vector Macro Instructions	C-28
C-13	String Instructions	C-29
C-14	Logical String Instructions	C-31
C-15	Non-Typical Instructions	C-32
C-16	Monitor Instructions	C-35
C-17	Register Designators	C-36
G-1	Predefined Symbols	G-1

INTRODUCTION 1

The CONTROL DATA STAR Assembler is a versatile, self-extending source language and language processor which runs under the control of the CONTROL DATA STAR® Operating System (OS). From the source language subprograms, the STAR assembler generates binary output (relocatable) acceptable for loading and execution by the central processor under STAR OS control.

The source language consists of mnemonic machine instructions, procedures, functions, and miscellaneous assembler directives. With the symbolic machine instructions, all hardware functions of the STAR computer system may be expressed symbolically.

Directives allow programmer control of the assembly process.

FEATURES

This assembly language makes efficient use of all computer resources and provides flexibility in program construction.

Features include:

Simple and consistent notation.

Procedure and function capability (provides many-for-one object code generation).

Conditional assembly capabilities for selective assembly

Set capability to define, reference, and extend lists of expressions

Attribute assignment for symbols and set elements

Mnemonic machine instructions define instructions to be generated. (Appendix C describes machine instructions.)

All existing assembler routines are re-entrant to permit simultaneous use by many users and location-independent for fast loading.

ASCII Code set compatibility

Assignment of relocatable and absolute location counters for use in address assignment.

Comprehensive listing of maps, diagnostics, etc.

PROCEDURES

Procedures are assembly time subroutines that provide extensive parameterization of source statements through conditional assembly and many-for-one object customized generation.

Procedures may be used for:

- Assembler instruction expansion
- Parameter checking, set generation, symbol redefinition
- Building a new language
- Saving parameters at assembly time
- Changing instructions dynamically
- Defining tables external to each routine

A source statement, consisting of a procedure name and parameters, calls a procedure. The assembler interprets the procedure and generates the equivalent STAR relocatable binary object code. Often used or standard procedure definitions may be placed in the user defined library.

Procedure and function definitions are groups of source statements interpreted by the assembler each time a procedure or function is referenced. A reference to a procedure definition appears in the command field of a statement; it may be likened to a macro call. A procedure is similar to a macro.

FUNCTIONS

Functions are assembly time subroutines used where common routines (which return a value) are desired. Functions and procedures are defined in a similar manner; a function reference is similar to that of a FORTRAN function reference. Unlike a PROC a function does not generate code but returns a value. A reference to a function can appear in the label, command, or operand field of an assembler statement. In general, a function cannot appear in the label field of a statement. Only the SYM function can be used in this manner.

SETS/SYMBOLS

The programmer can define and assign symbols to an address, single value, or set (list) of data. An entire set can be referenced by a symbol; each element of a set can be referenced by adding one or more subscripts to the symbol.

The assembler recognizes as operands simple and complete expressions containing any of a set of 21 operators. Elements of expressions can be symbols, constants expressed as integers, or real (floating point) values, according to convenience.

A unique method of symbol definition allows the value of an expression to be used as a symbol. An operand of a source statement also can be an attribute of an expression, such as type, size, etc.

ATTRIBUTES

An attribute is a property of an elementary item or expression. The assembler assigns attribute values (1-7) to all symbols and set elements. These intrinsic attributes are used by the assembler during syntax checks and expression evaluation. Through attribute referencing, the programmer can obtain information pertaining to set elements or expressions, such as:

Symbol as a character string

Mode

Memory section location

Definition level

Symbolic type

Size

Number of elements

The programmer also can assign extrinsic attributes that are not used by the assembler but can be referenced later or changed by the programmer.

The range of the extrinsic attributes is 8-127; i.e., the programmer may assign 120 extrinsic attributes. A list of intrinsic attributes, including possible values assigned by the assembler, appears in section 5. Methods of referencing intrinsic and extrinsic values and of assigning extrinsic values are given in section 5 (ATT directive) and section 4 (RATT function).

BASIC PROGRAM STRUCTURE

Source statements for an assembler program can be in one of two program areas: universal or subprogram. Non-executable code and statements that do not generate data can be entered in the universal area; however, all code can be written in the subprogram area with the exception of I/O directives and assembly control directives described in section 4. The Universal and Subprogram areas are described in section 2.

LOCATION CONTROL

STAR Assembler directives permit program code and data to be assigned to a maximum of 255 subprogram control sections. Each control section has a location counter to ease the programming task of segmentation. All code and data locations are relative to the beginning of the control section and the counters can be incremented by words, bytes, or bits.

ASSEMBLY PROCESS

The STAR Assembler is essentially a two-pass assembler; however, the number of passes depends on the existence of the subprogram area. If the assembler is called and only a universal area exists in the source program, only one pass is made. If a subprogram area exists, the following occurs:

First Pass	All statements are interpreted, values are assigned to symbols, and locations are assigned to each statement.
Second Pass	Externals and forward references are satisfied, data generation is accomplished, binary output and assembly listing are produced. Statements are interpreted during this pass and, if required, error and warning messages are assigned.

OPERATING SYSTEM

The STAR Assembler executes under control of the STAR Operating System, as described in appendix D.

CONFIGURATION

The requirements for executing the STAR Assembler on the STAR Computer System are the minimum required for the STAR Operating System.

EXECUTION

The assembler is called from the system library by an assembler job control command (META); see appendix D. Parameters in the command define files to be used during the assembler run, such as source statement files, listable output files and object code files.

STANDARD INPUT

The assembler source deck can be input from a standard card reader or a file, such as mass storage file, specified by the programmer. For a card file, input staging transfers the deck from the standard input card reader onto a mass storage file. The assembler interprets one source deck statement at a time.

PRINTER OUTPUT

The assembler produces printer output containing a listing of each source statement. Control directives provide options for obtaining a detailed listing. Errors detected by the assembler are noted on the listing. The output listing may include:

Source Program

Memory Map (Address Counter)

Generated Object Code

Diagnostics

Cross-Reference Listing

Assembler diagnostics, are listed in appendix F; the assembler listing format is described in appendix E.

EXECUTABLE OUTPUT

Upon programmer request, the assembler opens the user specified file to receive relocatable binary output acceptable to the STAR relocatable loader. When the assembler has completely processed the source deck, the programmer can call for loading and execution of the object program from that file. The loader links the newly assembled programs referred to by a new program.

ASSEMBLER ERROR DETECTION

Errors detected by the assembler are indicated on the listing by an error message preceded by a field of asterisks; each message occupies a full listing line.

SOURCE STATEMENT ERRORS

Source statement errors are listed after the statement containing an error. The count of the number of errors, and a list of the line and page number of statements with errors are included in the listing after every subprogram. Pass one errors are listed after the IDENT statement for the subprogram.

STATEMENT TERMINATING

A statement terminating error is indicated by any error message NOT preceded by WARNING or SYSTEM ERROR. On detecting such an error condition, the assembler discontinues processing the current statement and continues with the next sequential statement.

WARNING MESSAGE

Messages beginning with the word WARNING indicate a default was assumed for this error condition and statement processing continued. (The LISTING directive may be used to eliminate warning messages from the listing.)

HARDWARE OR ASSEMBLER ERRORS

All hardware or assembler error messages start with SYSTEM ERROR. They indicate a failure within the assembler; the assembly is aborted.

PROGRAM STRUCTURE 2

Assembler programs are written in modular form; they can consist of one or more subprograms (figure 2-1) which are linked and loaded together, and executed as a task. The source code for each program is assigned to assembler-defined program areas – universal and subprogram. These areas can contain procedures and functions which, for discussion purposes, can be considered subroutines. Each subprogram area can contain one or more code, data, and common sections. Each subprogram area produces a separate object module in the object file.

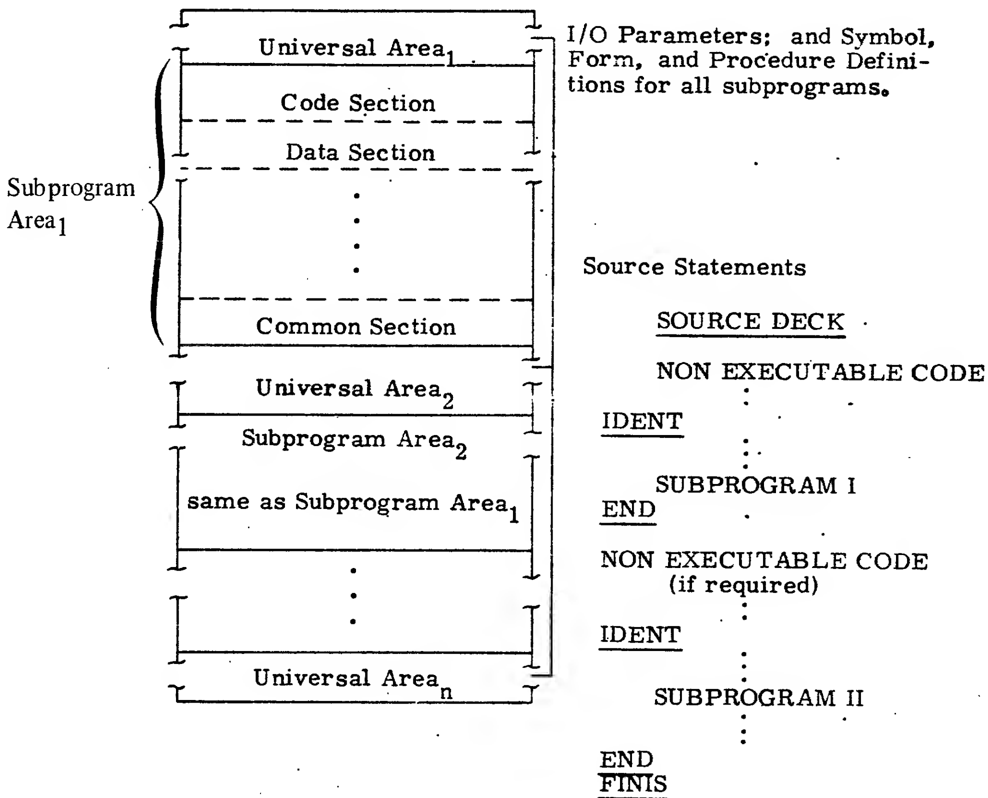


Figure 2-1. Program Structure

ASSEMBLER CODE CONVENTIONS

All code (except data and common section code) must be location-independent. Such code consists of a sequence of statements without virtual address references (relative references are permitted). This code is written to execute correctly from any location in virtual memory and combines the benefits of absolute code (fast loading) with that of relocatable code (can be loaded at any location).

Assembler code also must be re-entrant; it must never modify itself. Re-entrance permits the simultaneous use of the same code by more than one task in the user program. Re-entrant code is obtained by separating the code from data modified by the code.

Examples of location-independent and re-entrant code and the solution to some programming problems which result from these conventions are provided in appendix I.

PROGRAM UNIVERSAL AREA

The universal area is located before the first IDENT statement of the subprogram area (when a program consists of one subprogram) between the END statement of a subprogram area and the IDENT statement of the following subprogram area (when a program consists of more than one subprogram). Statements in the universal area specify input/output parameters and define symbols, procedures, functions, and sets to be referenced by statements following the subprogram areas which follow.

Procedures (section 4) are assembly-time (only) subroutines that generate customized code or data. Only one copy of a procedure is required regardless of how many times it is to be called within a program. Functions (section 4) are also assembly-time (only) subroutines normally used when common subroutines are required. Functions, unlike procedures, return a value, and cannot generate code.

The STAR assembler is essentially a two-pass assembler; however, in the universal area only one complete pass is made per assembly. Code or data cannot be generated in the universal area. Forward references (section 4) or statements which affect location counters (FORM references, MSEC) are not permitted. A reference to a symbol (appendix A) or set of elements (set name and a list of expressions) before it is defined is termed a forward reference. A reference to a numeric label is not considered a forward reference.

Definition level 1 is assigned to the universal area. All symbols defined in this area are assigned a definition level attribute of 1. All identifiers and names defined in a procedure or function and located in the universal area are assigned a definition level of 3 or greater depending on the nested call level. Each nest of a procedure or function call increases the definition level by 1. Symbol level definition and referencing is described at the end of this section.

SUBPROGRAM AREA

The subprogram area consists of statements between the IDENT and END directives. The subprogram area can consist of one or more user-specified memory control sections which can contain: code (code section) and associated data, data (data section) to be shared by more than one subprogram, and common data shared between two or more separately assembled programs. These memory control sections are assigned through the MSEC directive or by default as described in section 4.

In the assembler object file output, locations of code and data sections of a user program are non-contiguous. However at load time, these areas are linked through the register file; the STAR loader allocates contiguous locations for all common sections.

All assembler directives can be used in the subprogram area except the INPUT, OUTPUT, LISTING, IDENT, and FINIS directives. Forward references to non-redefinable symbols are permitted; however, forward references to function names, procedure names, form names and redefinable symbols are not permitted.

The subprogram area is assigned definition level 2; therefore, symbols not defined in a procedure or function are assigned a definition level attribute of 2 unless they are declared external. Symbols defined in a subprogram area procedure or function are assigned a definition level of 3 or greater depending on the nested call level of the procedure or function. Each nest of a procedure or function call increases the definition level by 1.

The subprogram area is two pass, therefore it does not permit nested forward references because an additional pass is required for the resolution of each nested reference.

CODE SECTION

The code section consists of the executable portion of the subprogram. Code section statements must be reentrant and location-independent and can contain read-only constants and instructions; external references and relocation references are not permitted. Read-only data is better placed in the data section, although it can be placed in the code section. When data is contained in the code section, it is not necessary to specify the start of the data section by MSEC directive.

DATA SECTION

The data section contains information unique to the user's program. The beginning of a data section is specified through the MSEC directive or through default. Relocatable and external references can be used in this area.

COMMON SECTION

This section consists of data which can be shared between programs assembled separately, but loaded together. This section is specified by the MSEC directive and contains a return address identifier. Variables, relocatable references, and external references are permitted here; however, symbols must not be declared as entry points.

LEVELS OF SYMBOL DEFINITION

The assembler recognizes 128 levels of symbol definition: external, universal, subprogram and 125 procedure/function call levels (table 2-1).

Symbols defined at a given level always are available at that level and all higher levels, but they cannot be referenced from a lower level unless they are made external. Symbols outside the assembly can be declared external through the EXTC or EXTD directives.

Within procedures, functions, or subprograms a dollar sign ($) appended to the symbol, when it is defined, changes the definition level of the symbol. At the subprogram level, the $ lowers the definition level to 1. When the $ is used within a procedure or function (or nested procedure/function), the definition level is lowered to 1 if the original procedure/function is called from the universal area; or it is lowered to 2 if called from subprogram area.

Table 2-1. Symbol Levels

Level Value	Meaning
1	Symbol is in universal area and available to all subsequent subprograms, functions, and procedures.
2	Symbol is in subprogram area and available to all procedures and functions called by the subprogram.
$\geqslant 3$	Symbol is in a function or procedure and available to all procedures and functions called by the procedure or function.

LEVELS OF SYMBOL REFERENCE

When a symbol is referenced, the assembler always searches for the symbol at the current level. If it is not found there, the assembler sequentially searches each lower level.

A symbol defined at both the originating call level and the current level, must have a $ appended to it when it is referenced to return the original call level value (either the universal area or subprogram area level value).

Symbols are defined through the SET, RDEF, or EQU directives. The RDEF directive in the following example illustrates the appended $. In this example, the symbol A is defined at the universal level, redefined at the subprogram level, and referenced at the subprogram level.

Example

```
Universal
area          A       RDEF        3          *A=3
(level 1)


Subprogram            IDENT
area
(level 2)     A       RDEF        2          *A=2
              B       RDEF        A+1        *B=3--REFERENCES LEVEL 2
              C       RDEF        A$+1       *C=4--REFERENCES LEVEL 1
```

For a second example illustrating the use of the $, see appendix I, ASSEMBLY TIME SQUARE PROCEDURE.

STATEMENT STRUCTURE 3

The STAR assembler language source program consists of a sequence of symbolic machine instructions, directives, and comment lines. Input may consist of a sequence of statements punched on 80 column cards or entered into a source file via a terminal display console or can be resident on mass storage in binary or source format.

The programmer can specify the begin, continuation, and end boundaries of each program statement through an assembler supplied INPUT directive or through default values. If the starting character position is not specified, a default value of 1 is assumed by the assembler. The assembler scans each statement as specified by the preceding INPUT directive or by default value.

Similarly, the programmer specifies the last character position (end-of-column) of each statement. The maximum value is 256; default value is 72. If a continuation for a field is specified before the end-of-column, the assembler scans the next line starting with default column 25 or a column specified by the INPUT directive. Continuation is specified by use of an ampersand (&).

Assembler statements can contain up to four fields; the fields must be separated by one or more blanks:

Label

Command

Operand

Comment

Each field can be as long as required. Should the length of a single field or combination thereof exceed 2^{47-1}, field continuation must be designated by inserting an ampersand (&) in the field to be continued.

Characters outside the statement boundaries are ignored, but the entire line image is listed by the assembler.

Table 3-1 describes statement format, including field restrictions.

Table 3-1. Statement Format

Format: Label, list command, list list *comments

Label Field	Command Field	Operand Field	Comment Field
Starts at: Begin column	Starts at: First non-blank character after Label field.	Starts at: First non-blank character after Command field.	Starts at: First non-blank character after Operand field starting with asterisk or any field starting with asterisk. If the first character of any field is an asterisk, characters following are considered comments.
Terminated by: Blank ‡ End-of-Record End Column	Terminated by: Blank ‡ End-of-Record End Column	Terminated by: Blank ‡ End-of-Record End Column	Terminated by: ‡ End-of-Record End Column
Format Description: numeric label, list Numeric label is optional. Optional list of elements separated by commas. Symbols (63 characters max). First character must be alphabetic Remaining characters must be A-Z, 0-9, or underscore. Set element references. Symbol creation function. Command field determines legal elements in list.	Format Description: command,list Command can be: Directive Machine instruction mnemonic Form name Procedure name Symbol creation function Optional list of elementary items or expressions separated by commas. List elements vary with the command.	Format Description: list Optional list is composed of elementary items or expressions separated by commas. List elements vary with command: For a directive, this field provides information required to perform a designated operation. For mnemonic machine instructions and procedures, list represents addresses, constant values, and expressions to be evaluated.	Format Description: Any ASCII character other than & is legal as a comment. & indicates continuation.

‡ Unique end-of-record/line character (#1F) at the end of each source statement. This character is inserted by the editor or card reader.

Examples

The following illustrates the use of all four fields and continuation:

```
7,A GEN,64,2 *GENE&
RATES A
```

Column 25

(Columns 1-3 contain numeric and symbolic labels.)

The following statement includes a blank label field:

```
 LOD DECK,T1
```

(There is no comment field in this statement.)

The following includes a command and comment:

```
 END *COMMENT
```

The following illustrates only a comment:

```
*THIS IS A COMMENT
```

As described in table 3-1, a label may consist of an optional numeric and symbolic list. If the numeric label is not used, the symbolic list starts in the start column specified by the INPUT statement or in default column 1.

DIRECTIVES 4

GENERAL

A programmer using the CONTROL DATA STAR Assembler directs the assembly of object code by using a set of commands called directives. Directives control the operation of the assembler in much the same way as machine language instructions direct the computer. Through the use of directives, a programmer can:

Define a symbol and assign a value or set of values to it for subsequent reference by the symbol.

Specify that a symbol referenced by the program being assembled is defined externally (perhaps by a program previously assembled) or that it can be referred to by some other program.

Conditionally repeat or skip source statements.

Assign up to 255 relocatable location control counters for use by the assembler in address assignment.

Generate code to be loaded and executed on the object computer. This process includes subdivision of each word to be generated into fields, and the assignment of values to the fields.

Identify a group of statements as a function, assign one or more names to it, and use the assigned name as a value in an expression such that the value varies according to parameters of the function reference.

Control the format and content of the assembly listing.

Terminate assembly of subprogram or group of subprograms.

Table 4-1 (at end of section 4) summarizes assembler directives. Examples illustrating the use of these directives are provided in appendix I.

INPUT/OUTPUT CONTROL

The following directives specify the format of assembler input and the type and format of assembler output.

INPUT

The INPUT directive specifies source input format to the assembler:

numeric-label   INPUT   p10,p11,p12   * comments

Usage

Numeric-label is optional

p10   Specifies starting column of input record. Default is 1; p10 must be greater than zero.

p11   Specifies last column to be processed. Default is 72; p11 must be greater than p10 plus p12.

p12   Specifies starting column of continuation records. Default is 25; p12 must be greater than zero.

More than one INPUT directive is allowed per assembly. This directive is permitted in the Universal area only. Any syntax error in this statement terminates assembly.

Example

```
INPUT ,80,25 *SCAN SPECS
```

Start scanning at column 1, default.

Scan the entire field length of 80 columns.

Start scanning continuation records at column 25.

OUTPUT

The OUTPUT directive requests an object deck output:

numeric-label   OUTPUT   p30   * comment

p30   Request for a debug symbol table in the object file. If p30 has a value of 1, the debug symbol table is included in the object deck produced by the assembler. For any other value, the debug symbol table is not produced.

The OUTPUT directive can be used only in the universal area, and only one directive per assembly is permitted. A syntax error in this statement terminates object deck creation.

Example

```
OUTPUT
```

An object deck is to be created and no Debug Symbol Table Dump is requested.

LISTING

The LISTING directive is used to request assembly listing options.

numeric-label LISTING p14,p15 * comments

p14 Value of 1 requests a cross reference list, including all address and EQU definitions and all references that occur after the definition line, for example:

B	EQU	A	line 1
A	EQU	4	line 2
C	EQU	A	line 3

The cross reference listing will indicate that A is defined on line 2 and referenced on lines 1 and 3. Default of p14 ≠ 1 indicates no cross reference.

p15 Value of 1 specifies that warning messages are to be omitted from the listing.

Syntax errors result in selection of default values. This directive is permitted only in the universal area. Only one LISTING directive is allowed per assembly.

Example

```
LISTING 1,1
```

A cross reference list is requested and warning messages are suppressed.

LIBP

The library file can include PROC and Function source statements and comments. Any other statements are syntactically checked but not processed. A LIBP file must be a physical, mass storage file. Tape libraries are not permitted.

The LIBP directive specifies library procedures and functions. A syntax error terminates this directive:

numeric-label LIBP,p13 list15 * comments

p13 Optional; 8-character symbol specifying the source file name.

list15 List of procedures or function names separated by commas. If list15 is not used, all procedures and functions on the file will be available; otherwise, only those specified will be available. The list of procedure/function names must appear in the order in which they occur in the LIBP file.

Up to ten library files may be specified, one per LIBP directive.

The LIBP directive is not allowed within a procedure or function definition, and must appear in the universal area.

The following is an example of defining system parameters in a library:

LIBP File xx

```
                PROC

GLOBALS         NAME
SYSTEM$         EQU     "STAR OS"
TAPES$          EQU     5
                 .
                 .
                 .
                ENDP
```

Main Program

```
                LIBP,XX GLOBALS
                GLOBALS                  *DEFINES SYMBOLS AT UNIVERSAL LEVEL
```

LISTING CONTROL

The following directives specify the format of the assembler listing.

SPACING

Selects the number of blank lines between listing lines:

numeric-label SPACING p28 * comments

p28 Integer constant value of 0, 1, 2, or 3 indicates the number of blank lines to follow each listing line. Default is zero.

Example

```
SPACING 2 *SELECTS TWO BLANK LINES
```

When a syntax error occurs the SPACING directive is ignored. A SPACING directive overrides any previous SPACING directives at this level.

EJECT

Specifies listing is to resume at the top of the next page. EJECT can be used at all levels.

numeric-label EJECT * comments

TITLE

Places a title of up to 64 characters at the top of all succeeding pages; it also causes a listing eject.

numeric-label TITLE p29 * comments

p29 "character string of no more than 64 characters"

Example

```
TITLE "ASSEMBLER LISTING" *NO COMMENTS
```

MESSAGE

Forces a character string or string expressions (maximum 128 characters) to an output listing; it overrides any active list control directives.

numeric-label MESSAGE p16 *comments

p16 "character string or string expression" to be entered on the listing; if greater than 128 characters, the string will be right truncated.

Example

```
MESSAGE "ADD_PHASE_COMPLETED"
```

NOLIST

Suppresses a listing until a list directive is encountered.

numeric-label NOLIST *comments

LIST

Restarts output listing previously suppressed by a NOLIST directive. The normal mode of assembly is LIST. This directive does not alter the DETAIL mode. When DETAIL mode is off, statements processed as part of procedures and functions are not listed.

numeric-label LIST *comments

DETAIL

DETAIL is used only at expansion time not at definition time. At call time, this directive causes a listing of all statements processed as part of procedures or functions. A DETAIL directive processed at any level initiates the listing for the current level and all lower levels, until a BRIEF directive is encountered. DETAIL does not initiate the LIST mode.

numeric-label DETAIL *comments

If a LIBP directive is encountered while in DETAIL mode, the Procedure or Function definitions contained in the specified file are not listed.

If at level 4 DETAIL is encountered and at level 5 BRIEF is encountered, only level 4 code will be expanded. If again at level 6 DETAIL is encountered, then level 6 code is expanded.

Example

```
                                                          INPUT   1,80
                                                        OUTPUT
                                00 000000000032      B    RDEF     50
                                                          IDENT
                                                          DETAIL
                                                          FUNC    NUMBER
                                                     SQUARE   NAME
                                                     AGAIN   NAME
                                                     RESULT   RDEF     NUMBER(1)*NUMBER(1)
                                                          ENDP    RESULT
                                00 000000000019      B      RDEF    25
                                                     D        GEN    SQUARE(B)
                                00 000000000271      RESULT   RDEF     NUMBER(1)*NUMBER(1)
01 000000000000 F           00000000 00000271             ENDP    RESULT
                                00 000000000032      C     RDEF    B$
                                                           GEN    AGAIN(C)
                                00 0000000009C4      RESULT   RDEF     NUMBER(1)*NUMBER(1)
01 000000000040 F           00000000 000009C4             ENDP    RESULT
                                                          END
```

See Example 5, Appendix I for an assembly of the above example without the DETAIL directive.

BRIEF

Prevents the listing of statements processed as part of procedures or functions (turns off DETAIL mode). The BRIEF directive does not initiate the LIST mode. The default listing mode is BRIEF.

numeric-label BRIEF *comments

ASSEMBLY CONTROL

The following directives define program boundaries to the assembler:

IDENT	(used at level 1 only)
END	(used at level 2 only)
FINIS	(used at level 1 only)

IDENT and END directives specify the beginning and end of a subprogram; FINIS specifies the end of source statements.

IDENT

numeric-label,symbol   IDENT   *comments

symbol   Optional name of object deck. This symbol is truncated to the first eight characters when the object deck is produced. The symbol is not defined (as a label) as a result of this statement.

END

numeric-label   END   p1   *comments

p1   Optional address identifier indicating a transfer address for object deck execution. This identifier must have appeared previously as an entry point name in an ENTRY directive. (See SUBPROGRAM LINKING.) For an example of the use of this symbol, see Appendix I, example 8.

This statement can be followed by another Universal Area and Subprogram area. This provides the user with two separate assemblies with one deck setup. However, the user must ensure that only one Universal Area includes an OUTPUT directive and that the last SUBPROGRAM area ends with a FINIS directive.

FINIS

numeric-label   FINIS   *comments

FINIS terminates an assembly and must appear in the Universal level. If FINIS is encountered in a subprogram area, the assembly aborts in pass 1.

CONDITIONAL ASSEMBLY

The user can specify the conditions which must be satisfied before a source statement or group of source statements can be assembled and the number of times these statements are to be processed.

RPT

Specifies the number of times a statement or delimited group of source statements, following the directive, are to be processed:

numeric-label,symbol RPT,p26 p27 *comments

symbol — Optional variable identifier, or expression evaluating to a variable identifier containing current repetition count. This identifier can be referenced and altered by the user. The initial value is always 1; it is incremented by 1 with each repetition of the succeeding source statements. Symbols can be re-used as shown in example 5.

p26 — Number of times succeeding statements are to be processed; p1 must be an integer constant or a variable or expression which evaluates to an integer constant. If the value of p26 is zero or a negative, the RPT statement skips to the statement following the numeric label p27; p26 cannot be a forward reference (a symbol or set element referenced before it is defined). The value assigned to p26 upon encountering the RPT loop cannot be changed. See example 2 below.

p27 — Forward numeric label of the last statement in the RPT loop; p27 must be an integer constant or a varaible or expression which evaluates to an integer constant, and not a forward reference. If p27 is negative or zero, an error message is given and the directive is ignored. Loop can be nested and have common termination statements (see example 3).

Example

Some directives contained in the following examples are described later in this section.

1.
```
A        RPT,8      1
A        RDEF       A+1
1        GEN        A
```

The above repeat is equivalent to:

```
A        RDEF       2
         GEN        2
A        RDEF       4
         GEN        4
A        RDEF       6
         GEN        6
A        RDEF       8
         GEN        8
```

2.

```
                                                          INPUT    1,80
                                                          OUTPUT
                                                          IDENT
                      00 00000000000A             A        RDEF     10
                      DEC              10         1        RPT,A     1
                      00 000000000005             A        RDEF      5
01 000000100000 F   00000000 00000001             1        GEN      I
01 000000100040 F   00000000 00000002
01 000000100080 F   00000000 00000003
01 0000001000C0 F   00000000 00000004
01 000000100100 F   00000000 00000005
01 000000100140 F   00000000 00000006
01 000000100180 F   00000000 00000007
01 0000001001C0 F   00000000 00000008
01 000000100200 F   00000000 00000009
01 000000100240 F   00000000 0000000A
                                                        END
```

3. Nested RPT's.

```
                                                          INPUT    1,80
                                                          OUTPUT
                                                          IDENT
                      DEC               6         A        RPT,6     1
                      DEC              11         B        RPT,11    1
01 000000100000 F   00000000 00000001             1        GEN      A,B
01 000000100040 F   00000000 00000001
01 000000100080 F   00000000 00000001
01 0000001000C0 F   00000000 00000002
01 000000100100 F   00000000 00000001
01 000000100140 F   00000000 00000003
01 000000100180 F   00000000 00000001
01 0000001001C0 F   00000000 00000004
01 000000100200 F   00000000 00000001
01 000000100240 F   00000000 00000005
01 000000100280 F   00000000 00000001
01 0000001002C0 F   00000000 00000006
01 000000100300 F   00000000 00000001
01 000000100340 F   00000000 00000007
01 000000100380 F   00000000 00000001
01 0000001003C0 F   00000000 00000008
01 000000100400 F   00000000 00000001
01 000000100440 F   00000000 00000009
01 000000100480 F   00000000 00000001
01 0000001004C0 F   00000000 0000000A
01 000000100500 F   00000000 00000001
01 000000100540 F   00000000 0000000B
01 000000100580 F   00000000 00000002
01 0000001005C0 F   00000000 00000001
01 000000100600 F   00000000 00000002
01 000000100640 F   00000000 00000002
01 000000100680 F   00000000 00000002
01 0000001006C0 F   00000000 00000003
01 000000100700 F   00000000 00000002
01 000000100740 F   00000000 00000004
01 000000100780 F   00000000 00000002
01 0000001007C0 F   00000000 00000005
01 000000100800 F   00000000 00000002
01 000000100840 F   00000000 00000006
01 000000100880 F   00000000 00000002
01 0000001008C0 F   00000000 00000007
                 .
                 .
                 .
01 000000101E00 F   00000000 00000006
01 000000101E40 F   00000000 00000006
01 000000101E80 F   00000000 00000006
01 000000101EC0 F   00000000 00000007
01 000000101F00 F   00000000 00000006
01 000000101F40 F   00000000 00000008
01 000000101F80 F   00000000 00000006
01 000000101FC0 F   00000000 00000009
01 000000102000 F   00000000 00000006
01 000000102040 F   00000000 0000000A
01 000000102080 F   00000000 00000006
01 0000001020C0 F   00000000 0000000B
                                                        END
```

4.

```
A          EQU              4
B          EQU              5
           RPT,A.GT.B       5
           .
           .
           .
5          RES,64           64*12
C          GEN              7,7
```

The above repeat acts as a skip-to statement:

```
C          GEN              7,7
```

because A is not greater than B.

5.

```
                                                                       INPUT     1,80
                                                                       OUTPUT
                                                                       IDENT
                                  DEC             10          A        RPT,10    1
01  0000000000000  F   00000000  00000001                     1        GEN   A
01  0000000000040  F   00000000  00000002
01  0000000000080  F   00000000  00000003
01  00000000000C0  F   00000000  00000004
01  0000000000100  F   00000000  00000005
01  0000000000140  F   00000000  00000006
01  0000000000180  F   00000000  00000007
01  00000000001C0  F   00000000  00000008
01  0000000000200  F   00000000  00000009
01  0000000000240  F   00000000  0000000A
                                  DEC             5           A        RPT,5     1
01  0000000000280  F   00000000  00000001                     1        GEN   A*A
01  00000000002C0  F   00000000  00000004
01  0000000000300  F   00000000  00000009
01  0000000000340  F   00000000  00000010
01  0000000000380  F   00000000  00000019
01  00000000003C0  F   00000000  00000005
                                                                       GEN       A
                                                                      END
```

GOTO

The GOTO directive requests a conditional skip of source statements:

numeric-label GOTO,p9 list14 *comments

p9 Must be symbol, set reference or expression with no forward references and must evaluate to an integer constant. p9 can specify the list elements to be selected; or it can be in the form of logical expressions, the validity of which determines whether a skip is executed. If true, p9 = 1 the first element is selected. If false p9 = 0 the GOTO is ignored. If p9 is omitted, list element 1 is selected; if p9 is a negative value or if the comma is used but p9 is blank, the GOTO is ignored. If the value of p9 exceeds the number of list elements, the last list element is selected.

list14 One or more elements indicating a forward numeric label to which the GOTO could skip. Each list element must evaluate to an integer constant value with no forward references.

Examples

The following examples contain directives described later in this section

1.
```
A       EQU               1
B       EQU               2
        GOTO,B.GT.A+A     2
        GEN               A
        .
        .
        .
2       GEN               B
```

In this example p1 is an expression the result of which is false; therefore, the next statement assembled after the GOTO is:

```
        GEN               A
```

2. In the following example the source for (a) and (b) was identical. However in (b) the statement "A RDEF 1" was not assembled.

a)

```
                                                    INPUT    1,80
                                                    OUTPUT
                                                    IDENT
                         00 000000000009      A      RDEF      9
                                                    GOTO,     19
                         00 000000000001      A     RDEF     1
01 000000100000 F    434F4D4D 41204953      19      GEN     "COMMA IS OPTIONAL"
01 000000100040 F    204F5054 494F4E41
01 000000100080 F    4C
                                                    END
```

b)

```
                                                    INPUT    1,80
                                                    OUTPUT
                                                    IDENT
                         00 000000000009      A      RDEF      9
                                                     GOTO        19
01 000000100000 F    434F4D4D 41204953      19      GEN     "COMMA IS OPTIONAL"
01 000000100040 F    204F5054 494F4E41
01 000000100080 F    4C
                                                    END
```

3.
```
           GOTO          5
           .
           .
           .
7,A        RDEF          3+A
5,A        RDEF          A+4
```

p9 is missing; therefore, the first list element is selected. The next statement assembled after the GOTO directive is:

```
5,A        RDEF          A+4
```

4. The same muneric labels can be re-used provided they are not within the range of a single GOTO operation.

```
C     EQU                 2
D     EQU                 1
B     EQU                 2
A     EQU                 6
      GOTO,A.GT.B+D       2
1     GEN                 1
2     GEN                 2
      GOTO,C.GT.D         2
1     GEN                 1
2     GEN                 2
```

results in:

```
                                                                   INPUT   1,80
                                                                   OUTPUT
                                                                   IDENT
                          00 00000000000002              C         EQU     2
                          00 00000000000001              D         EQU     1
                          00 00000000000002              B         EQU     2
                          00 00000000000006              A         EQU     6
                          DEC                 1          10        GOTO,A.GT.B+D     2
01 00000000000000 F     00000000 00000002                2         GEN     2
                          DEC                 1          50        GOTO,C.GT.D     2
01 00000000000040 F     00000000 00000004                2         GEN     4
                                                                   END
```

RPT AND GOTO PROCESSING

In functions and procedures, RPT and GOTO directives are processed at call time rather than at definition time.

RPT and GOTO ranges must be in the same level as the RPT and GOTO directives.

If a RPT directive is within the range of another RPT directive, the range of the inner RPT must be totally within the range of the outer RPT.

If GOTO directives are within the range of RPT, the GOTO can branch outside the range of the RPT. In this case, the RPT is terminated, but the repeat symbol maintains its current value for later use.

An RPT directive must not be the last statement of an RPT range.

SUBPROGRAM LINKING

Subprograms are linked through the directive entry (ENTRY) and external data and code (EXTD and EXTC). The user can reference, with a program, an address identifier defined in another program.

Since the programs might be assembled at different times, the address values of these symbols cannot be known at assembly time; therefore, certain symbols are declared as entry or external at assembly time. This declaration is noted by the assembler and placed in the object code. At load time, the loader must interpret entries and externals.

ENTRY

An entry is a symbol (address identifier) defined in the program which declares the symbol to be an entry point. It also can be referenced as an external from another program. An address identifier or variable identifier assigned a value with the EQU directive is defined as an entry through the ENTRY directive. This symbol is truncated to 8 characters.

numeric-label ENTRY list4 *comments

list4 One or more address identifiers or variable identifiers (defined by EQU directives) that are made available outside the subprogram and defined at the program level. This list can contain forward references.

This directive cannot be used in the universal area (level 1).

Example

```
QST     IDENT
        ENTRY           SQRT "DECLARED AS ENTRY
        .
        .
        .
SQRT    EX              #41*64,2 "THIS IS ENTRY POINT FOR SQRT
        .
        .
        .
        END
        FINIS
```

When a symbol is declared to be an entry, the symbol must appear in the label field of some statement within the program. The EX instruction in this example is a machine instruction, described in appendix C.

EXTERNALS

An external is a symbol (address identifier) referenced in a program which declares the symbol external, but which is defined (given an address via ENTRY directive) in a separate program. The loader links all externals and entries; after all routines are loaded, the loader places the virtual address of the symbol declared as an entry into every occurrence of that symbol provided in other subprograms declaring it as an external. The assembler provides two external directives:

EXTD Declares data address identifiers not defined within the subprogram in which they are referenced, but defined in a data memory section of some other subprogram.

EXTC Performs the same function as the EXTD directive except the external reference must be defined in the code memory section of some other subprogram.

The format descriptions for the EXTD and EXTC directives are similar; the general format for both is shown below, and exceptions are noted. The braces { } specify that either of the enclosed can be selected.

numeric-label,list6 $\left\{ \begin{array}{l} \text{EXTD} \\ \text{EXTC} \end{array} \right\}$,p32 list25 *comments

list6 Optional list consisting of one or more symbols separated by commas. Each symbol becomes an address identifier for the first full word generated by the directive. If data generation is not indicated, the assembler ignores these symbols and warning messages appear on the listing.

p32 (EXTD) Optional integer constant, or an expression or variable which evaluates to an integer constant. If p1 evaluates to integer constant zero or blank (null), a full word (aligned to a full word boundary) is generated for each symbol in list25. After loading, this word contains the address of a designated data entry point. If p32 evaluates to any other value, no data is generated, and any symbols in list6 are ignored. The length field is not altered by the loader and may be preset during assembly (see FORM).

Example

```
DESC     FORM      16,48
         EXTD,1    A          *NO WORD GENERATED

B        DESC      12,A
```

p32 (EXTC) Integer constant or expression or variable that evaluates to integer constant zero or blank (null), two full words (aligned to full word boundaries) are generated for each symbol in list25, after loading, these words will contain addresses of the designated code entry point (first word) and its associated data area (second word). If other than zero or blank (null) no data is generated and symbols are ignored.

list25 One or more symbols external to the program, separated by commas, and which are truncated to 8 characters.

EXTD and EXTC directives must not appear in a code memory section. For referencing external code or data address identifiers, only two operators are permitted + and -.

Example

EXTD:

```
EXTD,1*A                     no data generation

GEN (A+64*5)                 legal reference since operation is addition
```

EXTC:

1. The EXTC and ENTRY directives permit reference of an address identifier defined in another subprogram. (Machine instructions used in this example, EX and BSAVE, are described in appendix C.)

```
*SUBPROGRAM 1
    R_63        RDEF   #63*64        *S/R ADDRESS LOADED
    RTN         RDEF   #1A*64        *RETURN REGISTER
    DATA        RDEF   #1E*64        *DATA BASE (SUPPLIED BY LOADER)

         IDENT
    A           EXTC   SQRT          *DECLARES SQRT EXTERNAL
                MSEC   2             *CODE MSEC ADDRESS OF SQRT
                LOD    DATA,R_63
                BSAVE  RTN,R_63
                END
    *
    *SUBPROGRAM 2
    ABC     IDENT
            ENTRY       SQRT         *DECLARES SQRT AN ENTRY
            MSEC        2
    SQRT    EX
             .
             .
             .
           BSAVE        ,RETURN      *RETURN TO CALLER
           END
             .
             .
             .
           FINIS
```

2.
```
A   EXTC   B,C,D,E
```

Designates symbols B, C, D and E as external code address identifiers. Two full words are generated for each external symbol. A is defined as an address identifier pointing to the first full word generated.

SYMBOL AND SET DEFINITION, AND REFERENCING

Sets are normally defined through the use of the SET directive; however, they can be defined by the following statements and directives:

ENDP EXITP	Return a subset for a function call value
NAME	
Procedure Call	Can define up to 4 sets
Function Call	Can define up to 2 sets

SET

The SET directive assigns the label field symbol as the set name for a list of expressions, set names, set element references, or subsets. (Set element references and subsets are discussed later in this section.)

```
numeric-label,list23     SET     list24     *comment
```

list23 One or more variable identifiers expressions, set element references or set names separated by commas. The elements of this list are the set names for list24. If the list23 set name was defined previously by a SET or RDEF directive, the name is redefined as a new set list.

list24 Set elements separated by commas. It can include expressions, set names symbols, set element references, or subsets. Elements of list24 can include repetition and positional operators. Repetition operators can be nested; positional operators cannot. A positional operator can appear within a repetition if its value is 1. List24 elements assume the value defined during SET directive processing. To change the value of an element, the user must redefine the set list element. Also, the number of list24 elements can be extended by redefining the entire set list with a SET directive.

An empty list24 element is specified by two adjacent commas. Zero is the implied value and the mode of the element is null.

Symbols in list24 become copies of the original symbols. If a symbol name in list24 is redefined or changed in a statement following the set statement, the set list element is not changed.

```
                                                                  INPUT    1,80
                                                                  OUTPUT
                                                                  IDENT
                                    00 00000000000A      A        RDEF     10
                                    00 000000000005      B        RDEF      5
                                                         C        SET      A,B
                                    00 000000000014      A        RDEF     20
01 000000000000 F    00000000 0000000A                           GEN      .ELM.C
01 000000000040 F    00000000 00000005
01 000000000080 F    00000000 00000014                          GEN      SYM(ATT(C[1],1))
                                                         E        SET   1,[1,[2,[3]]]
01 0000000000C0 F    00000000 00000000                           GEN      E[4]
```

Examples

The following examples illustrate the rules for positional operators and null elements in set definition.

POSITIONAL OPERATOR

When a positional operator (:) is used, the value to the left of the operator specifies the set position assigned to the value to the right of the operator. All previous set positions between that occupied by the previously specified set element and the value to the left of the positional operator are null positions.

```
list-1          mnemonic              list-2
A               SET            5, 4:2, 3
             1st element             indicates the 4th element is the value 2
```

defines a set:

5, [null] , [null] ,2,3

This is equivalent to:

```
A               SET            5,,,2,3
```

where A [2] and A [3] are nulls.

```
A               SET            1,3:[4,3:5],2
```

defines a set:

1, [null] , [4, [null] ,5] ,2

where: A [1] = 1

A [2] = null

$$A[3] = \left\{ \begin{array}{lcl} A[3,1] & = & 4 \\ A[3,2] & = & null \\ A[3,3] & = & 5 \end{array} \right\}$$

A [4] = 2

Positional operators must appear in ascending order, left to right.

A SET 3:2,1=6 is illegal, and illegal positional operator is ignored by the assembler.

NULL ELEMENTS

A null element can be specified by use of a positional operator or by double commas:

1.

(specifies null)

```
         A          SET          ,2,,3
```

same as:

```
         A          SET          2:2,4:3
```

A [1], A [3] and A [5] are nulls and return a value of zero. The integers in [ ] specify the positional location of the elements referenced. Referencing elements A [2] and A [4] returns the values 2 and 3, respectively. Since A [5] is outside of the set a null is returned.

2.

```
                                                     INPUT    1,80
                                                     OUTPUT
                                                     IDENT
                                                B    SET      1,2,3,4,5
01 000000000000 F   00000000 00000001                GEN      B[1],B[2],B[3],B[4],B[5],B[6],B[10]
01 000000000040 F   00000000 00000002
01 000000000080 F   00000000 00000003
01 0000000000C0 F   00000000 00000004
01 000000000100 F   00000000 00000005
01 000000000140 F   00000000 00000000 }  B[6] and B[10] are null values
01 000000000180 F   00000000 00000000 }
                                                     END
```

REPETITION

To specify value, set name, etc., in succeeding positions within a set list, the user can specify a repetition factor for that element. Repetition is requested by an integer or an expression or variable which evaluates to an integer constant (specifying the number of times the element must be repeated) followed by the elements in parentheses.

```
     A          SET          5,3(2),2
```

is equivalent to:

```
     A          SET          5,2,2,2,2
```

Repetition can also be specified for subelements of an element in a set list.

```
     A          SET          5,2(3,4),2
```

is equivalent to:

```
     A          SET          5,3,4,3,4,2
```

```
     A          SET          ,2([1,2],3)
```

is equivalent to:

```
A          SET          ,[1,2],3,[1,2],3
```

[1,2] are subelements of set A.

REFERENCING SETS

A set reference can appear in label, command, or operand field lists and must not be a forward reference.

A set reference consists of a set name and the position of the desired set element enclosed in brackets [ ]. Should the user specify

```
D        SET        A,B,C
```

and desire the value "B", he would reference the set as follows:

```
GEN        D[2]
```

because B is the second element of set D.

Should the user desire the entire set then the set reference would be written as:

```
GEN     .ELM.D    which returns    A
                                   B
                                   C
```

A reference such as:

```
GEN        D
```

results in an error message

```
XX ILLEGAL DATA IN FORM/GEN IN OPERAND FIELD
```

ELEMENT AND SUB-ELEMENT REFERENCING

A set element and sub-element is referenced by writing the set name with following expressions that specify the ordinal location of the element or sub-element. A set element reference can be written in the field list portion of the label, command, or operand of any statement.

The elements of a set can consist of many sub-elements; which are specified as an element by enclosing them in brackets [ ]. e.g.

```
B          SET          5,[6,7]
```

The name of the particular set followed by expressions locate the desired elements or sub-elements.

set-name [expressions]

Sub-elements [6,7] comprise the second element of set B. These sub-elements can be referenced as follows:

```
GEN   .ELM.  B [2]     returns 6,7 - to obtain 6 and 7 .ELM. must preceed the element reference
GEN          B [2,1]   returns 6   - set B, element 2, sub-element 1
GEN          B [2,2]   returns 7   - set B, element 2, sub-element 2
```

The following would generate an error message, "ILLEGAL USE OF .ELM. OPERATOR IN OPERAND FIELD

```
GEN   .ELM. B
```

ASSIGNMENT

Values are assigned to a symbol by the Redefine (RDEF) and Equivalent (EQU) directives.

RDEF

Assigns the value and attributes of an operand field expression to the symbols specified in the label field. A symbol initially defined by this directive may be redefined using the same directive. Symbols defined by RDEF may not be forward referenced.

numeric-label,list5   RDEF   p3   *comments

list5   One or more variable identifiers, set element references, or set names separated by commas, that assume the value and attributes of p3.

p3   Any expression; p3 cannot be a set name. p3 cannot contain a forward reference to a statement that contains a forward reference. p3 cannot be a forward reference to a redefinable quantity (another RDEF or SET element).

```
A    RDEF    B
B    RDEF    C      Not Permitted
C    RDEF    1
```

If p3 contains a forward reference, the list symbol cannot be used in a statement that could affect the location counter. p3 cannot reference symbols declared external in EXTC or EXTD directives. e.g.:

```
A    RDEF    B
B    RDEF    1
     RES     #A*64*64     *RES IS DESCRIBED UNDER
                          *LOCATION CONTROL
```

Examples

A	RDEF	15	A has integer constant value of 15.
B	RDEF	@	B has address identifier value equal to the current location counter.
C	RDEF	A+3	C has integer constant value 18.
C	RDEF	C+2	C has integer constant value 20.
E	SET	3,5	
E[2]	RDEF	6	Redefines element 2 with a value of 6.
E	RDEF	2	Redefines set E to a variable identifier.

EQU

EQU assigns the value and attributes of an operand field constants, expression or variable to the symbols specified in the label field. A symbol defined by EQU cannot be redefined. Symbols defined by EQU may not be forward referenced.

numeric-label,list5 EQU p3 *comments

list5 One or more variable identifiers or single set element references separated by commas that take on the value and attributes of p3. List elements can be defined as entry points; however, in this directive, they must be defined as hexadecimal constants. If not a hexidecimal constant a mode error occurs.

List elements cannot be redefined.

p3 Any expression; p3 cannot be a set name or a redefinable quantity that is not yet defined and cannot contain a forward reference to a statement that contains a forward reference.

```
A       EQU     B
B       EQU     C        Not Permitted
C       EQU     1

A       EQU     B        Permitted (except in universal area)
B       EQU     1
```

If p3 contains a forward reference, the list symbol cannot be used in a statement that affects location counting.

p3 can contain references to symbols declared external with the EXTC or EXTD directives.

Examples

1.

```
C       EQU     1
D       EQU     2
A       EQU     D.LT.(C + 2)
B       EQU     A+E              *EQUIVALENT TO B EQU 4
E       EQU     3
```

2.

```
A       EQU     10
A       RDEF    12
```

Error: A-DOUBLY DEFINED

3.

```
A       SET     2,5
A       EQU     10
```

Error: ILLEGAL OPERAND OR PARAMETER

4.

```
A       SET     2,5
        GEN     .ELM.A           *GENERATES 2 AND 5

A       RDEF    10
        GEN     A                *GENERATES 10
```

DATA GENERATION

Data generating directives define data format and generate information to be placed in the object deck.

FORM

Defines a data generating format that specifies alignment and field size in bits.

numeric-label,list7 FORM,p4 list8 *comments

list7 One or more symbols separated by commas. Each symbol becomes a name used to reference the form.

p4 Variable or expression resulting in an integer constant representing the bit alignment for the current location counter when the form is referenced. Forward references are not permitted. If p4 is not included, a value of 1 is assumed. Any value is acceptable; however, 1, 8, 16, 32, 64, 128, and 256, and 512 are recommended.

list8 List of expressions, variables or integer constants, separated by commas. The value specifies the field size of the form in bits. The value must evaluate to or be an integer constant with no forward references. The fields specified in list8 can be repeated by using the repetition operator; repetition can be nested. Null elements are not permitted. These values specify field size in bits and can be any value.

Defining a symbol to be a form name does not restrict the use of that symbol as an address or variable identifier.

Example

```
WORD        FORM       24           1 field, 24 bits long, aligned to a bit boundary

WORD2       FORM,64    48           1 field 48 bits aligned to a full word boundary

2,CHARS     FORM,8     8,8,8,8      4 fields, each 8 bits aligned to a byte boundary

I           SET        8,8,48       Defines I as a set consisting of [8,8,48].

AA,INST     FORM,64    .ELM.I       3 fields, aligned to a 64-bit boundary. The form
                                    has two names.

A           FORM,32    4(8,16)      8 fields, aligned to a ½-word boundary
```

is equivalent to:

```
A    FORM,32     8,16,8,16,8,16,8,16

B    FORM,64*512   1,15,48          3 fields, aligned to a 512-word page boundary
          (virtual page size)

DESCRIPTOR     FORM,64     16,48    length (0-15) and address (16-48) of vector
                                    descriptor
```

FORM REFERENCING

A form reference generates data starting at the first bit after alignment is performed. The data is stored in the memory section containing the reference. Form references must not appear in a function or a procedure called via a function call.

numeric-label,list9 form-name list10 *comments

list9 Address identifiers, separated by commas; they assume the value of the current location counter after alignment is performed.

form-name Name of the form to be referenced.

list10 List of expressions, separated by commas. The value of each expression is placed into the field of the form. The position of the expression in list10 specifies the field destination for the value. The positional operator and repetition operator can be used with the expressions in list10. If list10 is longer than the number of fields in the form, the form fields are repeated, but alignment is not repeated.

Examples

```
                      Word Boundary
WORD        FORM,64     16,48
A           GEN         24
            WORD        1,A
                Call
```

Generates a full word aligned to a word boundary with the value 1, right justified and zero-filled, in bits 0 through 15; and the address of A right justified and zero-filled, in bits 16 through 63.

```
LABEL       WORD        "AB",@
```

Generates a full word with characters "AB" in bits 0 through 15 and the value of the current location counter (requested by use of @) after alignment. The value @ is right justified, zero-filled in bits 16 through 63.

```
            EXTD,1      A     *DEFINE A AS EXTERNAL WITHOUT
                               GENERATING DATA
D           WORD        3,A   *SET BITS 0-15 WITH 3 AND LOADER
                               WILL SUPPLY EXTERNAL DATA ADDRESS
                               OF A IN BITS 16 TO 63.
```

```
                     Byte Boundary
CHAR        FORM,8      8
B1          RDEF        "B"
            CHAR        2("A",B1),6:1}
```

Generates a byte string aligned to a byte boundary. The first byte contains "A", the second, "B", the third "A", and the fourth "B". Contents of the fifth byte is zero, and the sixth contains the value 1 right justified, zero-filled.

```
CHAR        FORM,8      8
I           SET         2("A",B1),6:1
            CHAR        .ELM.I
```

This example is equivalent to the immediately preceding example.

GEN

Generates data starting at the next aligned bit; data is stored in the memory section containing the directive. The GEN directive must not appear in a function or a procedure called by a function call or in the universal area of a program.

numeric-label,list12 GEN,p4,p8 list13 *comments

list12 Address identifiers, separated by commas. They assume value of the current location counter after alignment is performed.

p4 Must be an integer constant or a variable or expression which evaluates to an integer constant with no forward references; specifies alignment in the current location counter. Alignment is performed prior to data generation and applies only to the first expression in list13, p4 must be greater than zero and without forward reference. The default value is 1.

p8 Integer constant or a variable or expression which evaluates to an integer constant specifying the number of bits to be reserved for each expression in list13. It must be greater than zero and without forward reference. If p8 is not included, the mode and value size of each expression list13 specifies the number of bits to be reserved.

list13 List of expressions, separated by commas. The value of each expression is the data generated. The repetition operator can be used with expression in list13.

The rules for data generation specified in appendix A (see CONSTANTS) are applicable to the GEN directive.

Examples

```
A            GEN,64      -5
```

Generates a full word aligned to a full word boundary. Bits 0 through 63 contain the value -5 with sign extended.

```
B            GEN,,48     A
```

Generates 48 bits with bit alignment (specified by ,,). Bits 0 through 47 contain the address of **A**.

```
C            RDEF        "ABCDE"
             GEN,32      5,C,P"32"
```

Generates a full word aligned to a half-word boundary with the value 5 in bits 0 through 63; also generates a 5-character byte string containing "ABCDE" and a 2-character byte string containing the signed packed constant P"32".

```
GEN          D
•
•
•
```

Generates a full word with bit alignment (default size specification). Bits 0 through 15 contain zero; bits 16 through 63 contain the address of D.

```
2,D          GEN         B"10010"
```

Generates (5 bits) 10010 aligned to a bit boundary.

```
GEN,32,64*10             I"-2"
```

Generates 10 full words aligned to a half-word boundary; they contain -2 in binary integer form with sign extension.

```
GEN,64,128               10(B"10")
```

Generates ten 128-bit fields with the first field aligned to a full word boundary. Bit 126 of each field will be set.

ADDRESS AND LOCATION CONTROL

The code and data sections of an assembler subprogram are assigned to specific virtual memory areas. Code or data can occur in any memory section; however, externals are not allowed in MSEC code and entry points are illegal in a common MSEC. Code and data are assigned explicitly through the MSEC directive. Absence of an MSEC directive implicitly assigns code and data to the default data memory section IMEM. (IMEM is a reserved symbol assigned to a default MSEC.)

Each memory section has a unique relocatable location counter. The loader (not the programmer) determines the memory section where code or data is to be stored. Each location has the same relocation as the memory control that defined the location counter. Location counters are bit incremented; all memory addresses, therefore, are bit addresses.

An ordinal number is reserved for each location counter (each MSEC) sequentially in the order of memory control section definition. The STAR assembler permits up to 255 control sections (ordinals) in any combination. Ordinal 1 is reserved for the default IMEM, created when an IDENT statement is encountered. Any subprogram can use one or more memory sections; however, only one location counter can be active at any one time. The current location counter is designated by the last MSEC, ORG, EORG directive, or the default MSEC.

All address identifiers derive their value from the currently active counter and take on the same relocation as the current counter. The value of the current location counter can be altered by the following statements which can also define address identifiers:

GEN directive reference

RES directive reference

ORG directive reference

Procedure reference

Form directive reference

Machine Instructions

All data generated is stored in the currently active memory section. The following statements cause data generation:

GEN directive reference

FORM directive reference

Procedure reference (unless called from a function)

Machine instructions

The assembler interprets a reference to a memory control section name as a reference to the current value of its location counter. Use of @ (commercial "at") returns the value of the currently active location counter. The following example illustrates explicit specification of a memory section on a typical assembler printout.

```
                                                                   IDENT
02 000000000000                                                    MSEC  2
                                                                   ENTRY START
                         00 000000001000   A                       EQU   #40*64 * THESE REGISTERS CONTAIN
                         00 000000001040   B                       EQU   #41*64 * SOURCE ELEMENTS
                         00 000000001080   C                       EQU   #42*64 * CONTAINS RESULT VECTOR DESCRIPTOR
                         00 000000000014   N                       EQU   20     * LENGTH OF RESULT VECTOR "C"
                         00 000000000740   PSP                     EQU #10*64    *
                         00 000000000540   VITAL                   EQU #15*64    *** ENTRY SEQ  SEE APPENDIX K
                         00 000000000680   RTRN                    EQU #1A*64    *

02 000000000000 F   BE400000 00000001   COMMENCE                   EX    A,1     * VALUE 1 SOURCE
02 000000000040 F   78400041                                       RTOR A,B      * TRANSMITS VALUE 1 TO B SOURCE
02 000000000060 H   2A420014                                       ELEN C,N      * VALUE 20 ENTERED INTO LENGTH PORTION OF C DESC.
02 000000000080 F   DF000040 00410042                              INTERVAL A,B,C *CREATES VECTOR C
```

Specifies MSEC with an ordinal of 2

location counter (address counter)

Specifies boundary, (Full, Half, Character, Bit)

DEFAULT MSEC

The default MSEC is aligned to a double-word boundary and identified as IMEM. IMEM is classed as a data MSEC with an ordinal of 01 and cannot include monitor mode instructions.

Note: MSECs following the first MSEC 1 will not align to a boundary larger than double word even when specifically requested.

Example

```
INPUT
OUTPUT
IDENT
MSEC      1
   .
   .
   .
MSEC      1
GEN,64*512  10
        ↑
```

The last instruction tries to align to a page boundary, but it doesn't work

MSEC

This memory section directive defines a control section and makes it current. The MSEC directive can be used only in subprogram areas and in procedures not called by functions.

numeric-label,list17 MSEC p18,p19 *comments

list17 — Optional address identifier used to reference the memory section. The current value of the memory counter is returned upon reference. The address identifier on a MSEC 3 will become the name of the common block.

p18 — Optional integer constant or variable or expression which evaluates to integer constant 1, 2, or 3 specifying the kind of control section.

1	Data MSEC
2	Code MSEC
3	Common data MSEC

Default is 1.

p19 — Optional integer constant or variable or expression which evaluates to an integer constant indicating whether monitor instructions are permitted in the memory section.

1	Monitor instructions permitted
≠1	Monitor instructions not permitted

Default is monitor instructions not permitted.

Multiple code memory sections within the same program area are concatenated by ascending ordinal number to form one memory section. Each memory section is aligned on a word boundary after concatention. The same is true for multiple data memory sections.

The use of multiple common memory sections within one subprogram area requires a unique address identifier list17 for each common MSEC.

Examples

1.
```
A MSEC
```

Defines A as the name of a data memory section (default for p18 is data). Monitor instructions are not permitted (default for p19).

2.
```
A MSEC 1      " DATA MSEC
B MSEC 2,1    " CODE MSEC WITH MONITOR INSTRUCTIONS
C MSEC 3      " COMMON MSEC
D MSEC 4      " WARNING MESSAGE - THERE IS NO 4 OPTION TO P18--&
                DATA MSEC IS DEFINED VIA DEFAULT
```

3. The following example demonstrates a means of communicating with MSEC COMMON.

```
                    INPUT   1,80
                    OUTPUT

DATA_BASE           EQU     #1E*64
R_C_BASE            EQU     #20*64
C1                  EQU     #21*64
T1                  EQU     #22*64
T2                  EQU     #23*64
                   IDENT

                    MSEC    1                *DATA MSEC
C_BASE              EXTD    COMMON           *LOADER WILL FILL WITH
                                              ADDRESS OF COMMON BLK
                    MSEC    2                *CODE MSEC
                    ENTRY   START
START               LOD     DATA_BASE,R_C_BASE
                    LOD     R_C_BASE,T1      *T1=FIRST WORD OF COMMON
                    ES      C1,1
                    LOD     [R_C_BASE,C1],T2 *T2=SECOND
                    ADDX    T1,T2,T1
                    STO     R_C_BASE,T1
                    END
                     .  (OTHER SUBPROGRAMS)
     .               .
                     .
                    IDENT
COMMON              MSEC     3
                    GEN      1
                    GEN      2
                    END
                    FINIS
```

Loader places address of program START's data base in Reg #1E; i.e., points to C_BASE. Loader places in memory location C_BASE the address of the common block COMMON.

RES

Aligns the current location counter and adds to it the value of the expression (bit value) in the operand field. This directive can be used in the subprogram area and in procedures not called by functions.

numeric-label,list17   RES,p4   p25   *comments

list17   Optional list of address identifiers (separated by commas) their values are the values of the current location counter after alignment.

p4   Optional integer constant or variable or expression which evaluates to an integer constant specifying alignment in bits. Default is 1 (bit boundary). Any value may be selected; however, 1, 8, 16, 32, 64, 128, 512 are recommended.

p25   Optional expression with an integer constant value specifying the bit value to be added to the current location counter after alignment. Default is 0.

Examples

```
A RES,32        32*512
```

Reserves 512 half-words aligned to a half-word boundary.

```
L EQU      100
BYTE RES,8      8*L
```

Reserve 100 bytes aligned to a byte boundary.

ORG

Sets the location counter to a specific value; the memory section associated with the location counter then becomes active. The ORG directive can appear in a subprogram area or in procedures not called through functions.

numeric-label,list26   ORG   p21   *comments

list26   Optional list of address identifiers, separated by a comma. Each address identifier in the list assumes the value of the current location counter after the ORG is completely processed.

p21   Expression or variable which evaluates to an integer constant or integer constant value of an associated memory section ordinal. The bit value, p21, is the value that becomes the location counter of the memory section implied by the ordinal number. The current memory section becomes associated with the ordinal number.

If p21 has no ordinal number, the current location counter is set to the value of p21.

Examples

```
1.  A    MSEC
    B    MSEC
    C    ORG      A+64
```

Sets the location counter of MSEC A to that of its current value plus one full word.

```
2.  A    MSEC
    B    GEN,64     5
    C    MSEC
         ORG        B+64
```

Sets the current location counter of memory section A to that of relocatable address B plus one full word.

EORG

Sets the current memory section to the value of the memory section specified prior to the last MSEC or ORG directive. This directive can be used in a subprogram area or a procedure not called through a function.

numeric-label EORG *comments

Examples

```
1.  A   MSEC
2.  B   MSEC
3.      ORG   A
4.      EORG
```

In this example, the location counter is first set to the address of data memory section "A". In (2) a second data memory section is specified and the location counter is updated accordingly. The ORG directive sets the current memory section to A and updates the location counter to the address of MSEC "A". In (4) the current memory section is set to the value specified prior to the ORG directive; therefore, "B" is the current memory section.

```
A   MSEC
B   MSEC
    ORG   A            *specifies address of memory section A,
    ORG   @+512*64     *specifies current address plus 1 full page
    EORG
```

Sets B as the current memory section.

```
A    MSEC
B    MSEC
     EORG
```

Sets A as the current memory section.

ATTRIBUTE CONTROL

Extrinsic attributes are assigned, referenced, and changed by the user; attribute numbers may vary from 8 to 127. (Intrinsic attributes and the ATT function are described in section 5.)

An extrinsic attribute is assigned and changed with the RATT directive.

RATT

numeric-label,list21   RATT   list22   *comments

list21 — One or more address identifiers, variable identifiers, set element references and/or set names whose attributes are to be changed.

list22 — A list of elements, separated by commas; each has the form p1:p2.

- p1 — Is the attribute number; the value of p1 must be an integer constant, an expression or variable which evaluates to an integer constant greater than or equal to 8 and less than 128. An identifier can have up to 120 extrinsic attributes. Within one RATT directive, each p1 entry must be unique. See example 3. Also, p1 values must be in ascending order.
- p2 — Is the value of the extrinsic attribute. The value of p2 must evaluate to a constant with no forward references.

The RATT directive cannot be used with the intrinsic attributes (1-7).

Examples

1.

```
A          RATT     8:5,9:#10
```

then:

```
ATT(A,8) IS 5
ATT(A,9) IS 16
```

See section 5 for a description of the ATT directive.

2.

```
                                      INPUT   1,80
                                     OUTPUT
                                      IDENT
                                  B     SET     5,"ABC",6,"DEF"
      00 000000000001             C     EQU   1
                                  B[2]    RATT      9:C,10:B[1]
                                      END
```

The 9th attribute of B[2] is 1; the 10th attribute of B[2] is 5.

3.

```
                                               INPUT   1,80
                                              OUTPUT
                                               IDENT
                                           B     SET     5,"ABC",6,"DEF"
               00 000000000001             C     EQU   1
                                           B[2]    RATT      9:C,9:B[1]
****   IMPROPER USE OF POSITIONAL OPERATOR, (:) IN OPERAND FIELD
                                               END
```

REFERENCING ATTRIBUTES

Attributes are referenced through the ATT function described in section 5.

PROCEDURES

A procedure (PROC) is an assembly time subroutine that normally, can be used to generate code. This type of procedure is called in-line; and when it is called, it returns generated code to the location from which it was called. Procedures can be defined in the universal or subprogram areas. Forward references are permitted only in those PROC's called from the subprogram area or from a lower level. When a procedure is called, all identifier names defined in the procedure are assigned to level 3 or greater depending on the nest level of the call. At call time, if a referenced symbol is not found in a procedure, the preceding levels are searched. Each time a procedure is called and code is returned, the object code increases proportionally because only one copy of the code will exist.

PROC's should be written in a generalized form which allows the internal definition to produce concise code.

WRITING A PROCEDURE

In writing a PROC, the programmer performs the following steps:

Defines what is to be accomplished.

Writes a definition such that a change to the PROC will include a change to others affecting it.

A procedure definition starts with a PROC directive and ends with an ENDP directive. The statements and directives within these limits are referred to as the statement body. Unless explicitly stated in the description of a directive, the directive can be used in the procedure definition in the subprogram area.

Example

PROC Definition

```
P1   PROC,P2    P3
     .
     .
     .
AA   NAME       *SPECIFIES AN ENTRY POINT TO THE PROC.
     .
     .
     .
     ENDP
```

Statement Body

The following applies to all Procedures.

- Procedures can be defined in the universal area at level 1 or in the subprogram area at level 2. When defined in the universal area, the PROC can be referenced from any universal or subprogram area that follows the definition.
- Procedure definitions may not be nested.
- Procedures can be referenced from any level.
- The definition of a procedure must precede a reference to it.
- Procedures defined in the subprogram area are lost when the END directive is processed.
- Procedures called through the use of a function must not contain any statements that could affect location counters.
- Procedures cannot be redefined.
- Symbols defined within a procedure are local to the procedure in which they are defined: the symbols are lost upon exit from the procedure. These symbols can be made available outside the procedure by appending a $ to them. On encountering the $, the assembler checks the call level for symbol definition, provided the procedure was called previously.
- Depending on the area from which the original call was made, procedures can define symbols in the universal or subprogram area when a $ is appended to the symbol.
- To reference a symbol in the universal or subprogram area that is also defined in the procedure, append a $ to the reference.
- Procedures can reference symbols defined at all lower levels, if the symbol is not also defined at the current level.
- Procedures can contain forward references to symbols defined within the procedure if the procedure is called from the subprogram or lower level.
- Procedures can include more than one NAME directive (entry point).

- A name within a PROC can call another name in the same PROC. Also, a name can call itself.
- Procedures are recursive to 128 levels.
- If two procedures within a library file have the same name and the name used is a PROC call the assembler will issue a diagnostic "MULTIPLE DEFINED SYMBOL".

PROC

Declares the start of a procedure definition:

numeric-label,p22   PROC,p23   p5,p6   *comments

p22 — Optional symbol that becomes the set name for the list of numeric labels and symbols that appear in the label field of the procedure reference statement. This set name is made available to this procedure when the procedure is called.

p23 — Optional symbol that becomes the set name for the list of expressions, set element references, and symbols that appear in the command field after the procedure name in a procedure reference statement. This set name is made available to this procedure when it is called.

p5 — Optional symbol that becomes the set name for the list of expressions, set element references, and symbols that appear in the operand field of the procedure reference statement. This set name is made available to this procedure when it is called.

p6 — Optional symbol that becomes the set name for the set list that appears in the operand field of the NAME directive. This set name is made available to this procedure when it is called.

Example

```
L_SET     PROC,C_SET     O_SET,N_SET
SUM       NAME           1,2,3
          GEN            L_SET[1]+C_SET[2]+N_SET[3]+O_SET[1]
          ENDP
```

This PROC uses all four sets.

NAME (PROCEDURE)

Defines a procedure name and the entry point of the procedure. This directive is processed when it is defined; statements following the NAME directive are processed when the procedure is called. Any number of NAME directives can be used in a procedure definition.

This directive, with some variation, is used in a function definition and described in that context under NAME (FUNCTION).

numeric-label,p7 NAME,p33,p20 list16 *comments

p7 A symbol that becomes the procedure name. This symbol is entered in the command field of a procedure reference (call).

p33 Optional integer constant, its bit value is the boundary for alignment of the current location counter when the procedure is called. If p33 is missing, no alignment is performed.

p20 Optional integer constant. If the value of p20 is 1, the symbols in the label field list of the procedure reference (call) remain undefined. If p20 is zero, > 1, or blank all symbols in the label field list of the call are defined as address identifiers. The value of each address identifier equals the value of the current location counter after alignment.

list16 An optional list of set elements which must be completely definable when the procedure is defined. Forward references are not permitted, and any symbols in this set list must be defined in the universal or subprogram area. The set name for this set list is the p6 symbol defined in the PROC directive.

Example

```
        PROC       B,A
        .
        .
ABLE    NAME,64    5,"ABCD",P"-25"
        .
        .
        ENDP
```

ABLE is the entry point to the procedure. When the procedure is called, the current location counter is aligned to a 64-bit boundary. When the procedure is called, the set A consists of the 3 elements: 5, "ABCD" and P"-25".

ENDP (PROCEDURE)

Terminates a procedure at definition and call time. With some variation, it is used to terminate a function definition and is described in this context under ENDP (FUNCTION).

numeric-label ENDP *comments

PROCEDURE REFERENCE

A procedure can be called (referenced) at any level through a procedure reference statement containing the procedure entry point name in the command field. During a call, parameters specified in the label, command, and operand fields can be passed to the procedure. A PROC must be defined before it can be called, and nesting can occur to a depth of 125_{10}. A procedure referenced through a function cannot contain a statement that affects a location counter. A summary of the relationship of the PROC directive, NAME directive, and procedure reference is illustrated in figure 4-1.

numeric-label,list18   p7,list19   list20   *comments

list18   Optional symbols, separated by commas and passed as parameters to the PROC definition directive. These symbols are defined as address identifiers, unless the p20 parameter in the called name (NAME directive) prohibits definition.

p7   Procedure entry point name that appeared in the label field of a NAME directive in a procedure definition.

list19   Optional list of set elements passed as parameters to the procedure.

list20   Optional list of set elements passed as parameters to the procedure.

List19 and list20 may consist of set names, set elements, subsets, symbols, and expressions. The repetition operator and positional operator also can be used in list19 and list20.

A user can insert a STAR instruction mnemonic or a directive name in a PROC call. The assembler checks the user-defined table before checking the internal definition table, thereby permitting redefinition of an instruction or directive. Once the user redefines an instruction or directive, however, the internal definition in the area redefined (universal or subprogram area) cannot be accessed.

PROCEDURE REFERENCE TERMINATION, EXITP

This directive terminates a procedure reference before the ENDP directive is encountered. More than one EXITP directive is permitted in a procedure or function. With some variation this directive is used to terminate a function reference and is described in this context under EXITP (FUNCTION).

numeric-label   EXITP   *comments

PROCEDURE REFERENCE FUNCTION FLOW

The following example illustrates how the assembler handles a procedure reference:

```
Procedure Definition  { LF       PROC,CF        OF,AF
in Universal Area     { CALL     NAME,A,LFSU    ARGU
                                 ENDP
                                 IDENT
                                 .
                                 .
                                 .
Procedure Call        { R_1,R_2  CALL,S_1,S_2,S_3   T_4,T_3,T_2,T_1
in Subprogram Area    {
                                 .
                                 .
                                 .
                                 END
```

Although the PROC call is defined in the universal area, it is called in the subprogram area and assigned a level of 3.

When the PROC statement is encountered, the assembler scans for a name line and ENDP directives. All other statements are checked for syntax errors.

When a procedure is called, a copy of the label, command, and operand sets in the procedure reference statement are passed to the PROC definition.

If the call is made in the universal area, all parameters and procedures must be defined before the call is made; since the assembler makes only one pass through the universal area. For a procedure call is in the subprogram area, it is not necessary to define all parameters prior to the call because two assembler passes are made through this area.

The sets passed are copies of the originals; therefore, the only method of changing elements in the original set is by appending a $ to the label in the label field of the PROC. When the sets are passed, as specified in the previous example the following argument results.

PROC Definition Symbols	Associated Call Parameters
LF	R_1,R_2
CF	S_1,S_2,S_3
OF	T_4,T_3,T_2,T_1

In addition to the three sets that can be passed at call time, the argument set exists as part of the operand field on the NAME line. Since a PROC definition can have more than one NAME line an argument set can exist for each. At any one time, the only applicable argument set is that associated with the called NAME directive.

Examples

When the PROC is entered it is possible to generate code/data. For additional examples see appendix I.

The following examples illustrate a PROC used to redefine a symbol in the command field.

```
                                               PROC,CF           OF
PROC           NO_GEN                          NAME
DEFINITION     SYM(ATT(CF[1],1),1)             RDEF              OF [1]
1                                              ENDP

               A_X                             RDEF              #9B
CALL                                           NO_GEN,A_X        "ONE"
```

At call time the value #9B is passed to CF and "ONE" is passed to OF. Since CF and OF represent a set, any reference to the set, even though each contains only one element, must be written [1]. The brackets specify set reference, and 1 specifies the first element.

```
                                   PROC,CF           OF
PROC           NO_GEN              NAME
DEFINITION     A_SET$              SET               25:0
2              I                   RPT,ATT(CF [1] ,7)   100
               100,A_SET [I]  RDEF                   OF [I]
                                   ENDP

CALL           B_SET               SET               6,"BIT",#9,X"4",20:0
               C_SET               SET               1,2,3
                                   NO_GEN,C_SET      B_SET
```

Of the three possible sets that can be passed to the PROC definition, two are passed.

The command and operand fields have only one set element. The defined C_SET has three elements and the B_SET has 20. Each C_SET element is a 40-bit integer constant. The defined B_SET is comprised of:

Element	Attribute
6	48-bit integer constant
"BIT"	Character string
#9	48-bit hexadecimal constant
X"4"	Hexadecimal string constant
15 null elements	

The 20th element is a 48-bit integer constant of 0.

The A SET is defined to have 25 elements: the first 24 are null elements; the 25th element is a 48-bit integer constant of 0.

This set name is also available at the call level after the PROC has been exited.

Since the NAME line has no parameters, no alignment is required and label field symbols are defined.

The repeat directive is set initially to 3. The seventh attribute of CF [1] returns the number of elements passed to the command field set. This value specifies the number of iterations of the repeat loop. The last statement in the repeat loop is at label 100.

The first occurrence through the loop redefines the value of the first element of the A_SET to be equal to the first element of the B_SET.

The second and third elements of the A_SET are redefined during the second and third iterations of the RPT loop.

The following are examples of procedures used for data generation:

```
A         PROC,B    C,D
          .
          .
JOE       NAME,64   5
          GEN       A[1]
          GEN       B[1]
          GEN       C[2]+D[1],C[1]

          IDENT
          .
          .
AA1       JOE,3     "AA1_1",5
```

This procedure call to JOE is the same as writing:

```
AA1       RES,64    0
          GEN       @
          GEN       3
          GEN       10,"AA1_1"
```

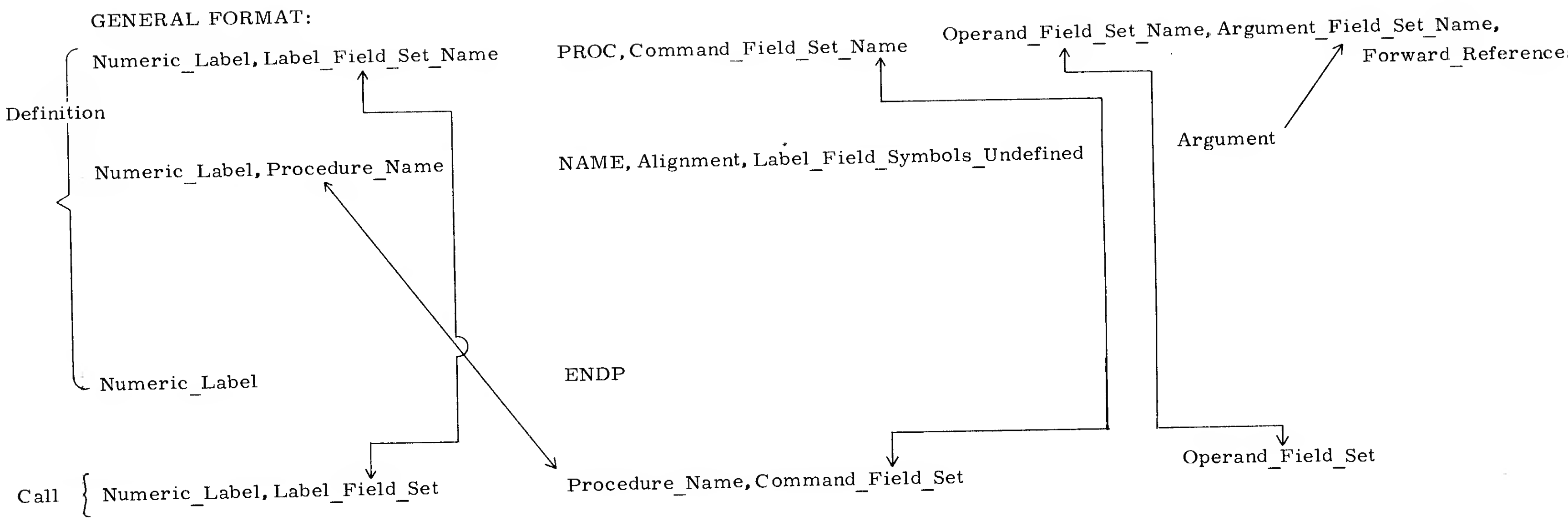


Figure 4-1. Association of Procedure Definition and Reference Elements

FUNCTIONS

Functions are assembly time subroutines normally used where common routines are desired. Unlike procedures, which are used for code/data generation or symbol redefinition, functions return a value to their place of reference.

FUNCTION DEFINITION

A function definition starts with a FUNC directive and ends with an ENDP directive.

```
                       FUNC
Statement   {  A_2     NAME     Entry point used in a function reference
Body        {          .
            {          .
                       ENDP
```

The statement body can consists of assembler statements other than the following:

OUTPUT	FINIS	PROC
LIBP	GEN	ENDP
END	FUNC	RES

When the assembler interprets a FUNC directive, it scans the succeeding lines of source code until a NAME directive is encountered. The scan lines are evaluated then but not processed; diagnostics are produced if a syntax error is encountered. Comments, are permitted between the FUNC and NAME directives. Lines between the FUNC and NAME directives are not processed at call time. Also:

- Functions defined in the universal area are at level 1. They are available to all subprograms.
- Functions defined in the subprogram area are at level 2 and are not available after the END directive is processed.
- Definition nesting is not permitted.
- Definitions must precede any reference to a function and cannot be redefined.
- Forward references are not permitted.
- More than one entry point (NAME directive) is allowed.
- Symbols defined within a function are not available outside the definition area unless a $ is appended to them.
- A symbol defined at or below the function call level can be referenced within the function, provided a $ is appended to the symbol at the definition level. When function calls are nested, the $ returns the search to the original call level (level of the first function call within the nest group). If the symbol is not defined at that level, the assembler drops back one level at a time until the definition is found. The same method is used by the assembler when a symbol referenced in an unnested function call is defined at a level lower than that of the call.

- A name in a Function can call another function. Also a name can call itself.
- Functions are recursive to 128 levels.
- If two functions in the library have identical names and if either is called an error message is generated. "MULTIPLY DEFINED SYMBOL".

FUNC

Declares the beginning of a function definition.

numeric-label FUNC p5,p6 *comments

p5 Optional symbol that becomes the set name for the list of expressions, set element references, and symbols that appear as the parameters in the function reference. This set name is made available when the function is called.

p6 Optional symbol that becomes the set name for the operand field set of the NAME directive. This set is made available when the function is called.

Examples

```
FUNC        A,B
.
.
.
ENDP        A[1]*B[1]
```

The parameter set name is A, and the set name for the set list on the NAME directive is B.

NAME (FUNCTION)

The NAME directive defines a function name and specifies the entry point of the function when it is called. This directive is processed only when it is defined and can be used only within a function or procedure definition.

numeric-label,p7 NAME list16 *comments

p7 Symbol that becomes a function name; it is used to call the function.

list16 Optional list of set elements: all set elements must be completely definable when the function is defined. Forward references are not permitted. Any symbols in this set list must be defined in the universal or subprogram area. The set name for this set list is the p6 symbol that appears in the FUNC directive.

Example

```
            FUNC        B,A
            .
            .
            .
ABLE        NAME        5,"ABCD",P"-25"
            .
            .
            ENDP
```

ABLE is an entry point name for the function. When the function is called, the set A consists of the three elements 5, "ABCD" and P"-25".

FUNCTION REFERENCES

A function is referenced by a function name. The function reference includes the function name assigned in the label field of the referenced function definition and associated parameters (see figure 4-2). A function reference can be made from any command or operand field. The parameter set in a function can contain a subset.

p7 list11

p7 — Function entry point name that appeared in the label field of a NAME directive in a function definition.

list11 — Optional list of set elements passed as parameters to the function.

ENDP (FUNCTION)

This directive terminates a function.

numeric-label ENDP p2 *comments

p2 — Optional expression or subset; p2 applies only to function definitions and is ignored if used in procedures. The value of p2 is returned as the value of the function call.

If p2 is not specified a null value is returned.

EXITP (FUNCTION)

This directive terminates a function reference before the ENDP directive is encountered. More than one EXITP is permitted in a function.

numeric-label EXITP p2 *comments

The rules for EXITP are the same as for ENDP. The value returned from the function can be any expression or subset. The function need not return a value.

Examples

For additional examples see appendix I.

```
1.                          FUNC      A,B
                            .
                            .
                ADDER       NAME      5
Definition                  .
                            .
                            EXITP     (A[1]+A[2])*B[1]
                            .
                            .
                            ENDP
                            IDENT
                            .
                            .
                B           EQU       6
                            .



Call                        GEN       ADDER(B,2)
```

This GEN with a function call is equivalent to:

```
                    GEN       40
```

2.

```
                    FUNC      PP
                    .
                    .
          A         NAME
                    .
                    .
          B         SET       5,2
                    .
                    .
                    ENDP      [B,PP[1]]
                    .
                    .
                    IDENT
                    .
                    .
          C         SET       A(3),4
```

This function call is the same as:

```
          C         SET       [[5,2],3],4
```

If the statement:

```
          D         SET       .ELM.A(1)
```

were entered in this example, the result statement would be:

```
          D         SET       [5,2],1
```

3.

```
                    FUNC      A,D
                    .
                    .
          CHAR      NAME      "REG_"
                    .
                    .
                    ENDP      D[1].CAT.A[1]
                    IDENT
                    .
                    .
          B         RDEF      CHAR("FULL")
```

This RDEF with a function call is equivalent to:

```
          B         RDEF      "REG_FULL"
```

GENERAL FORMAT:

```
            Numeric_Label                      FUNC    Call_Set_Name, Argument_Field_Set_Name
                                                                      Comments
                                                 .
Definition                                       .
                                                 .
            Numeric_Label, Function_Name       NAME    Argument               Comments
                                                 .
                                                 .
                                                 .
                                                 .
            Numeric_Label                      ENDP    Evaluated_Or_Subset_Returned_As_Value
                                                                      Comments

                      Call   Function_Name(Call_Set)
```

Figure 4-2. Association of Function Definition and Reference Elements

SUMMARY OF DIRECTIVES

The following tables provide the format of each assembler directive, its purpose, and the level at which each can be used. Symbols used to specify parameters, P, and lists items, L, are described in the table 4-2.

Table 4-1. Summary of Directives

General Format: numeric-label,list-1 name,list-2 list-3 * comments

Type	Name/Level	Format	Purpose
I/O	INPUT/1,2,n	numeric-label INPUT p10,p11,p12 *comments	Specific source input format.
	OUTPUT/1	numeric-label OUTPUT p30 *comments	Specifies object deck output format required.
	LISTING/1	numeric-label LISTING p14,p15 *comments	Specifies assembly listing options.
	LIBRARY/1	numeric-label LIBP p13, list-15 *comments	Specifies use of library procedures and functions.
Listing Control	NOLIST/1,2,n	numeric-label NOLIST *comments	Suppresses listing until assembler encounters LIST directive.
	LIST/1,2,n	numeric-label LIST *comments	Resumes listing suppressed by NOLIST.
	BRIEF/1,2,n	numeric-label BRIEF *comments	Suppress listing of statements part of procedures or functions.
	DETAIL/1,2,n	numeric-label DETAIL *comments	Lists all procedure and function statements.
	SPACING/1,2,n	numeric-label SPACING p28 *comments	Selects single, double, or triple spacing.
	EJECT/1,2,n	numeric-label EJECT *comments	Resumes listing from top of page.
	TITLE/1,2,n	numeric-label TITLE p29 *comments	Causes a listing eject and places specified character string at beginning of all succeeding pages.
	MESSAGE/1,2,n	numeric-label MESSAGE p16 *comments	Places a character string on the output listing.

Table 4-1. Summary of Directives (Cont'd)

Type	Name/Level	Format	Purpose
Assembly Control	IDENT/1	numeric-label, symbol IDENT *comments	Specifies beginning of subprogram area.
	END/2	numeric-label END p1 *comments	Specifies end of subprogram area.
	FINIS/1	numeric-label FINIS *comments	Specifies end of all source statements; terminates assembly.
Conditional Assembly Control	Repeat/1,2,n	numeric-label,symbol RPT,p26 p27 *comments	Specifies number of times source statements are to be processed.
	GOTO/1,2,n	numeric-label GOTO,p9 list 14 *comments	Specifies conditional skip of source statements.
Subprogram Linking	ENTRY/2,n	numeric-label ENTRY list-4 *comments	Specifies address ID's and variable ID's defined by EQU directives, which can be referenced by other subprograms.
	EXTD/2,n	numeric-label,list–6 EXTD,p32 list–25 *comments	Lists data address identifiers defined with ENTRY directive in data MSEC of another subprogram.
	EXTC/2,n	numeric-label,list–6 EXTC,p32 list–25 *comments	Performs above functions for code address identifiers.
Symbol and Set Definition	SET/1,2,n	numeric-label,list–23 SET list–24 *comments	Assigns label field symbol as a set name for list 24 contents.

Table 4-1. Summary of Directives (Cont'd)

Type	Name/Level	Format	Purpose
Assignment	Redefine/1,2,n	numeric-label,list-5 RDEF p3 *comments	Assigns or reassigns value and attributes in operand field to symbols in label fields.
	Equivalence/1,2,n	numeric-label,list-5 EQU p3 *comments	Assigns value and attributes in operand field to symbols in label field. After a value is assigned, symbol cannot be redefined.
Data Generation	FORM/1,2,n	numeric-label,list-7 FORM,p4 list-8 *comments	Specifies form name and defines data generating format by specifying alignment and field sizes in bits.
	Form Reference/2,n	numeric-label,list-9 form name list-10 *comments	Specifies generation of data from expressions in list 10 into field of form specified by form name referenced. (Form name is specified by FORM directive.)
	Generate/2,n	numeric-label,list-12 GEN,p4,p8 list-13 *comments	Specifies generation of data starting at next aligned bit.
Location Control	Reserve/2,n	numeric-label,list-17 RES,p4 p25 *comments	Aligns current location counter and adds value in operand field to counter.
	Memory Section/2,n	numeric-label,list-17 MSEC p18,p19 *comments	Defines control section and specifies it as current.
	Origin/2,n	numeric-label,list-26 ORG p21 *comments	Sets implied location counter to specified value. Activates memory section containing statement.
	End Origin/2,n	numeric-label EORG *comments	Sets current memory section to preceding memory section specified prior to the last MSEC or ORG directive.

Table 4-1. Summary of Directives (Cont'd)

Type	Name/Level	Format	Purpose
Attribute Control	Reference Attribute/1,2,n	numeric-label,list-21 RATT list-22 *comments	Adds or changes extrinsic attributes of identifiers.
Procedures and Functions	Procedure/1,2	numeric-label,p22 PROC,p23 p5,p6 *comments	Declares start of procedure definition.
	procedure reference/1,2,n	numeric-label,list-18 p7,list-19 list-20 *comments	Calls procedure and passed parameters to it.
	Function/1,2	numeric-label FUNC p5,p6 *comments	Declares start of function definition.
	function reference/1,2,n	p7(list-11)	Calls function and passes parameters to it.
	NAME/1,2	numeric-label,p7 NAME,p33,p20 list-16 *comments NOTE: p33 and p20 apply only to PROC's	Defines function/procedure names and entry points.
	†ENDP/1,2,n	numeric-label ENDP p2 *comments	Terminates procedure or function; parameter p2 is used only with functions.
	†EXITP/1,2,n	numeric-label EXITP p2 *comments	Terminates a procedure or function before END definition. (More than one EXITP allowed in procedure or function.) Parameter p2 is used only with functions.

†p2 applies to functions only.

Table 4-2. STAR Assembler Directive Parameters

Designator	Description
p1	Address identifier used to indicate a transfer address for object deck execution. Must have appeared as an entry point name on ENTRY directive.
p2	Optional expression or subset for function definitions; it is ignored in procedures.
p3	Any expression; it may not be a set name.
p4	Bit value for alignment of current location counter.
p5	Optional symbol that becomes the set name for the list of expressions, set element references, and symbols appearing in the operand field of the reference statement.
p6	Optional symbol that becomes the set name for the set list appearing in the operand field of the NAME directive that is the entry point.
p7	Symbol that becomes a function/procedure name, it can be in the command or operand field list of directives or instructions.
p8	Value indicating number of bits to be reserved for each expression in list 13.
p9	Indicates what list 14 element is to be selected.
p10	Beginning column of source code.
p11	Last column of source code.
p12	Continuation column of source code.
p13	Character symbol specifying the name of the source file for procedures or function definitions.
p14	Default: no cross reference. 1 Cross reference listing is desired. ≠ 1 No cross reference list.
p15	Warning messages are to be omitted from the listing.
p16	Character string of 128 characters or less to appear on output listing, overriding any active listing control directives.
p17	Optional symbol that becomes the memory section name.

Table 4-2. STAR Assembler Directive Parameters (Cont'd)

Designator	Description
p18	Optional integer that indicates usage restrictions. Default is 1 1 Data MSEC 2 Code MSEC 3 Common MSEC
p19	Optional integer constant permitting monitor instructions in this memory section. Default or value > 1 or < 1; no monitor instructions allowed. 1 Monitor instructions are allowed.
p20	Optional value of integer constant. Default or value > 1 or < 1 – all symbols in label field list of call will be defined as address identifiers. 1 Symbols appearing in label field list of procedure call will remain undefined.
p21	Any expression that has an integer constant value or a value that has a single memory section ordinal number associated with it. The bit value becomes the location counter of the memory section implied by the ordinal number. The current memory section is associated with the ordinal number.
p22	Optional symbol that becomes the set name for the list of symbols appearing in the label field of the procedure reference statement.
p23	Optional symbol that becomes the set name for the list of expressions, set element references, and symbols appearing in the command field after the procedure name of the procedure reference statement.
p25	Optional integer constant; must be a positive bit value, that is added to the current location counter after alignment.
p26	Indicates number of times succeeding statements are to be processed (if the symbol value is not altered within the repeat loop).
p27	Identifies a forward numeric label on the statement that is to be the last line repeated.
p28	Indicates number of lines to skip after each line listed (0,1,2, or 3).

Table 4-2. STAR Assembler Directive Parameters (Cont'd)

Designator	Description
p29	Character string of 64 characters or less to be printed at the top of succeeding pages.
p30	If set to 1, requests debug symbol table dump.
p31	Two-digit hexadecimal number specifying the ID of the source file for a procedure or function definition.
p32	Integer constant; if it evaluates to 0 or null, a full word (EXTD) is generated for each symbol in operand list. After loading, it will contain address of designated data entry point. For EXTC, two full words are generated; contains address of entry points and data area.
p33	An optional integer constant; the bit value is the alignment for the current location counter when the procedure is called. Default is alignment on bit boundary.
list1	Usually, address identifiers and set element references or variable identifiers and set element references.
list2	Consists of elementary items and expressions.
list3	A list of elements separated by commas; made up of elementary items and expressions.
list4	Address identifiers or variable identifiers defined by EQU directives that are made available outside the subprogram and defined at the program level.
list5	One or more variable identifiers, set element references, or set names separated by commas, that assume the value and attributes of p3.
list6	Address identifiers that are external to the subprogram.
list7	One or more symbols, separated by commas; each symbol becomes the form name used to reference the form.
list8	Expressions, separated by commas, whose values specify the field sizes of the form in bits. Must be integer constants.
list9	Address identifiers, separated by commas; the address identifiers assume the value of the current location counter after alignment.
list10	A list of expressions, separated by commas. The value of each expression is the data that goes into the form field.
list11	Optional list of set elements are passed as parameters to the function. Parentheses are not optional.
list12	Address identifiers separated by commas.

Table 4-2. STAR Assembler Directives List (Cont'd)

Designator	Description
list13	Expressions, separated by commas; the value of each expression is the data to be generated.
list14	Elements for which the values indicate forward numeric labels the GOTO can skip.
list15	Procedures or function names separated by commas.
list16	Optional list of set elements. All set elements must be completely definable when the procedure or function is defined.
list17	Optional list of address identifiers, separated by commas, which assume the value of the current location counter after alignment.
list18	Optional symbols defined as address identifiers, provided the parameter on the called NAME line does not indicate they must not be undefined.
list19	Set names, set elements, subsets, symbols, or expressions passed as parameters to the procedure.
list20	Set names, set elements, subsets, symbols, or expressions passed as parameters to the procedure.
list21	One or more address identifiers, variable identifiers, set element references, and set names for which attributes are to be changed.
list22	Elements, separated by commas, of the form N1:N2. N1 Attribute number N2 Value of extrinsic attribute.
list23	One or more variable identifiers, set element references, or set names, separated by commas, to become set names for the set list24.
list24	Set elements (expression, set name, set element reference, or subset) separated by commas.
list25	One or more symbols, separated by commas, external to the subprogram.
list26	Optional list of address identifiers that assume the value of the current location counter after ORG is processed.

ASSEMBLER PROVIDED FUNCTIONS AND PROCEDURES 5

The functions and procedures described in this section are provided as part of the assembler for use during program assembly. Functions and procedures described are:

Conversion functions

Symbol Creation functions

Attribute functions

NOPH procedure

SHORTBR procedure

NOTE

> Any symbol defined which is the same as a function name or assembler-provided function name, may override the function when a call to it is made; therefore results are unpredictable.

CONVERSION FUNCTIONS

Conversion functions provide the programmer with a means of changing a value from one constant form to another.

Function Call:

function-name (expression)

Table 5-1 lists the current assembler functions.

Table 5-1. Conversion Functions

Function Name	Function Performed
ITOC	Convert an integer or hex constant to an integer value represented as character string constant. Leading zeros are suppressed.
HTOC	Convert an integer or hex constant to a character string constant represented as a hexadecimal value.
PTOI	Convert a packed constant to an integer constant.
ZTOP	Convert a zoned constant to a packed constant.
DTOP	Convert an integer string constant to a packed constant.
XTOD	Convert a hex string constant to an integer string constant.
ITOF	Convert an integer or hex constant to 64-bit floating point.
BTOD	Convert a bit string constant to an integer string constant.
F32F	Convert a 32-bit floating point value to 64-bit floating point value.
FF32	Convert a 64-bit floating point value to a 32-bit floating point value.
ZTOC	Convert a zoned constant to a character string constant.
PTOZ	Convert a packed constant to a zoned constant.
ASSM(p1)	Return an integer constant depending on the value of p1.
	P1 = 1 Current value of error count.
	P1 = 2 Current value of warning count.

NOTE

For an example of the ITOC and HTOC function, see appendix I.

SYMBOL CREATION FUNCTION

The symbol creation function removes the quotes enclosing the first argument. It is used to convert character strings to symbols and to generate symbols. Symbols created by the SYM function will be entered into the symbol table.

SYM (p1,p2,p3)

p1 An expression that evaluates to a character string, ABC etc. Forward references are not permitted. No restriction on the characters in p1. i.e. # , - ,+, . . .

p2 Optional. When equal to 1 specifies the inclusion of a $ appended to the symbol, ie., symbol is at call level.

p3 Optional level number for explicit symbolic control.

Example

The following example illustrates symbol creation for use in the label field of a PROC statement. (A second example using the function appears in appendix I.)

```
P                           PROC
CALL-BY-NAME                NAME,,1
SYM(ATT(P[1],1),1)          RDEF    10
                            ENDP
A                           CALL-BY-NAME
```

This call statement results in the equivalent of the following statement:

```
A$          RDEF          10
```

In the SYM (ATT(P[1],1),1) statement:

(P[1],1) Requests the first sub-element of set A which is A; the result is as specified by attribute 1 which specifies the expression for use as a symbol.

,1) Specifies a dollar sign be appended to the symbol.

The following example illustrates the use of p3:

```
1           IDENT
SYM("A")        EQU 1           *SYMBOL A AT LEVEL 2
SYM("A",1,1)    EQU 2           *SYMBOL A AT LEVEL 1
     GEN         A              *GENERATES 1
     GEN         SYM("A",1,1)   *GENERATES 2
     END
```

ATTRIBUTE FUNCTION

Attributes may be intrinsic or extrinsic. The use of extrinsic attributes and the RATT directive are described in section 4.

INTRINSIC ATTRIBUTES

The attribute function followed by the attribute number is used to return the value of the specified attribute. The value returned provides information about the symbol referenced in that function. Intrinsic attributes, are listed below with the significance of values that can be returned when the attribute is used in an ATT reference.

ATTRIBUTE 1

Symbol as a character string – returned value equals the symbol or set element as a character string in quotes. A null character string is returned if there is no symbol, e.g.; if:

a) A GEN 5
   then
   ATT(A,1) returns "A"

b) B SET A
   then
   ATT(B[1],1) returns "A"

ATTRIBUTE 2

Mode – returns the mode of the expression as an integer constant.

Mode	**Value**
No value	0
Absolute address	1
Relocatable address	2
External address	3
Integer or hexadecimal constant	4
Hexadecimal string constant	5
Bit string constant	6
Character string constant	7
Real constant	8
Packed decimal constant	9
Zoned decimal constant	10
Integer string constant	11
Null element; element of a set list is not defined	12

ATTRIBUTE 3

Memory Section Ordinal Number – returns the ordinal number (integer constant) of the memory control section under which the address identifier is defined. A zero is returned if there is no ordinal.

ATTRIBUTE 4 — Definition Level – returns the definition level as an integer constant.

Definition Level	Value
Universal	1
Subprogram	2
Procedure/Function	≥ 3

ATTRIBUTE 5 — Symbolic Type – returns the symbolic type as an integer constant.

Symbolic Type	Value
Undefined	0
Redefinable identifier	1
Identifier not redefinable	2
Set name	3
Not an identifier (it is an expression or literal)	4

ATTRIBUTE 6 — Value Size – returns an integer constant indicating the number of bits needed to contain the value of the item.

ATTRIBUTE 7 — Number of Elements – returns the number of elements in the named set as an integer constant. If not a set, the value zero is returned.

```
B       SET     2,3,6,7,4
        GEN     ATT(B,7)   Returns a value of 5.
```

ATT

The implicit attribute of a symbol or a set element is its value. The value attribute of a symbol is synonymous with the symbol; no further notation is needed to obtain that information.

Example

```
A          RDEF          10

           GEN           A
```

The use of the A in the GEN statement returns the value attribute which is 10.

The attribute function is used to obtain attributes other than the value attribute.

The ATT function returns the value of the indicated attribute. (intrinsic or extrinsic).

ATT(p1,p2)

p1 — The symbol, symbol creation function, or the set element reference of which the attribute is to be retrieved.

p2 — An expression with an integer constant value specifying the attribute to be returned.

Unpredictable results may occur if extrinsic attributes are referenced before they are defined.

Examples

```
1.  A          GEN          5
```

The address identified A has the following attributes:

ATT (A,1) is A

ATT (A,2) is 2 (assume default MSEC)

ATT (A,3) is 1 (assume default MSEC)

ATT (A,4) is 2 (assume statement was in subprogram area)

ATT (A,5) is 2

ATT (A,6) is 48

ATT (A,7) is 0

```
D          GEN                 5
A          GEN                 D
           ATT(A,6) =          48
A          RDEF               "0"
           ATT(A,6) =          8
A          RDEF              I"25"
           ATT(A,6) =          8
A          EQU                 25
           ATT(A,6) =          48
A          SET 5,Z"+12"        12
           ATT(A[2],6) =       16
```

2.

```
                                              A        SET    1,2
01 000000000000 F  00000000 00000002                      GEN    ATT(A,7)
01 000000000040 F  00000000 00000000                      GEN    ATT(A[1],7)
                                                       END
```

Referencing a set element returns a null.

ASSEMBLER PROVIDED PROCEDURES

The following commands are provided for user convenience. They are alternatives to existing commands with preset values, qualifiers, or default values.

NOPH Used for alignment; no code generated. Half-word NO-OP can be used when aligning EXTD or EXTC generation in a data MSEC.

SHORTBR ADDRESS is equivalent to:

$$\text{BAB,BR} \begin{Bmatrix} \text{BRF} \\ \text{BRB} \end{Bmatrix} \text{,address}$$

For a description of the BAB mnemonic instruction see STAR HARDWARE Reference Manual.

GLOSSARY

Absolute Address

1. An address permanently assigned by the machine hardware to a particular storage location.
2. A pattern of characters that identifies a unique storage location without further modification. Synonymous with Machine Address. (See Virtual Addressing for Absolute Address).

Address

All addresses are 48-bit quantities containing enough information to reference a specific bit.

Address Identifier

A designator given to an execution time entity, such as a program point.

Assemble

To prepare an object language program from a symbolic language program by substituting machine operation codes for symbolic operation codes and virtual addresses for symbolic addresses.

Assembler Defined Program Areas

Source code for each assembler program is assigned to one of two assembler defined program areas:

Universal Area is used for I/O specification; symbol, procedure, function, and set definition.

Subprogram Area contains executable program statements.

Assembler Directives

The symbolic assembler directives control or direct the assembly processor in the same manner that machine instructions direct the central computer. Directives are represented by mnemonics.

Assembler Language Processor

A language processor that accepts words, statements, and phrases to produce machine instructions.

Assembly listing

A printed list presenting the logical instruction sequence. Included is symbolic source notation and actual object notation in hexadecimal form established by the assembly process. Relative virtual addresses of the assembler generated code are provided also.

Attribute

Characteristics of a symbol such as word size, mode of representation (hexadecimal, octal, etc.) The two attribute types are: intrinsic (1-7) - predefined. Extrinsic (8-120) - user defined.

Base Address

Address defining the origin or reference point of operands or results. It may be modified by offset or index to determine the desired address.

Byte

An 8-bit quantity, the address of the left most bit is always a multiple of 8.

Broadcast Constants

A 32- or 64-bit * 1 vector element used in some vector instructions to transmit the same vector element repeatedly. Broadcast or normal element is selected by machine instruction qualifiers.

Conditional Assembly

A feature of the STAR assembler that allows the user to dictate whether statements should be assembled or not. The user can achieve conditional assembly with the GOTO and RPT directives.

Control Vector (CV)

Base address of control vector is contained in Z field of vector and vector macro instructions. Control vector determines how many C elements are stored during execution of vector machine instructions and determines which pairs of A and B elements are compared during execution of Vector Macro instructions. Use is specified in an instruction by Z designator $\neq 0$, in which case, Z designator becomes the CV base address.

Elementary Item

A self defining component of an expression.

Entry

Symbol (address identifier), defined in the program that declares the symbol as an entry and can be referenced from another program.

Entry Point

Label of a source statement where execution or processing can begin.

Expression

Series of values, symbols, and functions connected by mnemonic or symbolic operators as required to cause computation.

External Symbol

A symbol (address identifier) referenced in the program that declares the symbol external but defined (given an address value) in another program.

Form Identifier

Designator identifying a form definition.

Forward Reference

A label referenced in the operand field that has not been previously defined.

Function

Assembly time subroutine normally used where common routines are desired. Functions return a value to the point of reference.

Function/Procedure Identifier

Designator for entry points defined within a function or procedure.

Half-word

A 32-bit quantity, the address of the leftmost bit always is a multiple of 32 (decimal).

Label

Labels may be numeric or alphanumeric. Alphanumeric labels comprise the label list of the statement format; they must start with a letter (maximum size 64 characters).

Location Counter

Counter assigned to each memory control section. They are incremented in bits and specify the bit location of code and data sections of a user program.

Location Independent Code

A sequence of statements containing no addresses. Such code is written to execute correctly from any virtual address without modification.

Memory Control Section

A specific area is user's virtual memory to which code and data can be assigned. Each MSEC is assigned an ordinal number. A maximum of 255 MSEC's can be specified in a user program.

Code MSEC can contain code and data. Data in this area is assigned to a specific user subprogram.

Data MSEC can contain information unique to a user's program.

Common MSEC can contain data that may be shared between programs assembled separately but loaded together.

Mnemonic Instruction

Use of symbolic notation in place of actual machine code. A mnemonic instruction must be translated to actual operation codes by assembler procedure references.

Normalizing a Number

The coefficient is shifted left until the sign bit does not equal the bit immediately to its right. The exponent is reduced by one for each left shift.

Offset

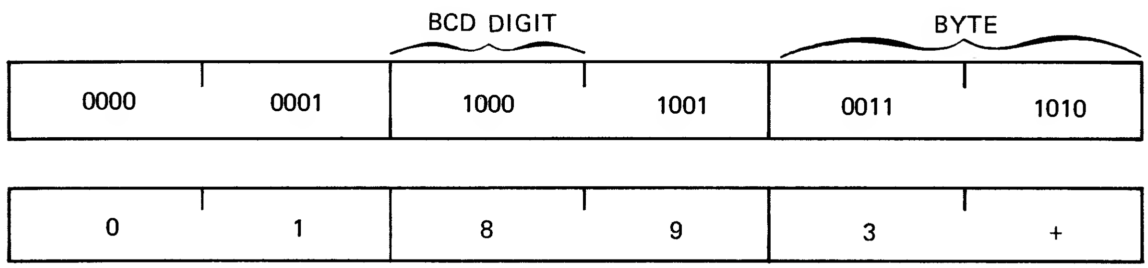
Number used to modify the base address of operands in vector and some non-typical instructions. May be half-words or words (determined by number of bits in operand up to ± 2^{15-1}).

Order Vector (OV)

Denotes non-significant elements in vector field. Generated by COMPARE instructions and used by COMPRESS instructions to generate sparse vector. Number of ones in order vector determines field length of sparse vector operands. A filled result order vector terminates sparse vector instructions.

Packed BCD Format

This format is used for decimal arithmetic. Two BCD digits are contained in each byte and the sign is right justified.



PACKED BCD FORMAT

Pre-defined Symbols

Symbols with special meaning to the assembler when used in the command field of an assembler statement.

Procedure

A subset of source statements meeting a specific purpose that can be repeatedly referenced to generate parameterized code.

Qualifiers

Symbols to indicate sub-operation of the function code specified by an instruction mnemonic.

Re-entrant Code

Code that never modifies itself. This type of code was used in writing this assembler to allow several users to employ the same assembler programs simultaneously.

Register File

256 registers of 64 bits each used for instruction and operand addressing, indexing, field length counts; source or destination of operands for register instructions. Addressed by 8-bit instruction designator .

Set

A collection of related elements having a common name. An element may be a set (a subset of a set). A reference to an element consists of the set name followed by one or more integers enclosed in brackets [ ] indicating the location of the element.

Source Program

A program written in assembly language that must be translated into machine language before it can be executed.

Sparse Vector (SV)

Vector field contracted by removing the non-significant elements to conserve storage space and calculating time. Positional significance of the elements is retained by an order vector for each sparse vector.

Statement

An instruction to be interpreted by an assembler.

Subscript

One or more integers enclosed by brackets [ ] used to specify a particular element in a set.

Subprogram

A part of a program determined by the IDENT directive (start) and terminated by an END directive.

Unary Operator

An operator such as the sign of a value (+ or -) that applies to one operand only, rather than causing addition or subtraction.

Vector (VT)

As used in the matrix algebra, a 32 or 64 x n array of elements. Maximum size is 64 bits x 65,536 words. Operates on ordered scalar contained in operand fields, rather than single operands.

Virtual Memory

A conceptual extension of main storage achieved by hardware technique which permits storage address references beyond the physical limitation of main storage. Virtual addresses are equated to real addresses during program execution.

Variable Identifier

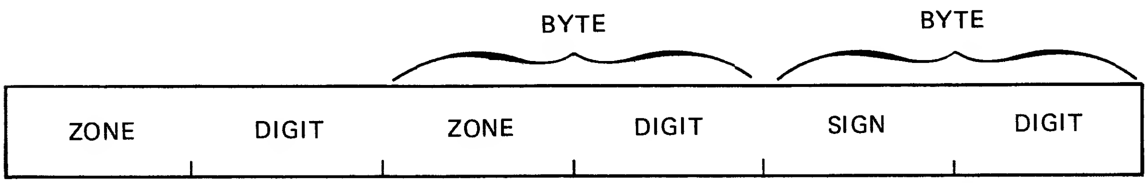
Designation of a single translation time value.

Word

A 64-bit quantity. The address of the leftmost bit is always a multiple of 64 (decimal).

Zoned BCD Format

Input/output operations use zoned format; one BCD digit is contained in each byte. Sign is leftmost 4 bits of rightmost byte. Leftmost 4 bits of all other bytes is called the zone. Instructions are provided for packing and unpacking decimal numbers so they may be changed from zoned to packed format and vice versa.



ZONED BCD FORMAT

ELEMENTARY ITEMS A

The basic representation of data for the assembler is an elementary item; it may be a delimiter character, symbol, variable identifier, constant, operator, etc. This appendix describes all elementary item types that can be used with the STAR assembler and provides examples of each type.

Table A-1 contains a complete list of the STAR character set. Subsequent paragraphs describe the type and use of these characters. A list of the operator characters and a description of their use in formulating expressions is provided in Appendix B. Delimiters are listed in table A-2, and Special Characters that have an implied meaning to the assembler are listed in table A-3.

Table A-1. STAR Character Set

Hex	Character	Punch	Hex	Character	Punch
20	ƀ space	no punch	41	A	12-1
21	!	12-8-7	42	B	12-2
22	" quote	8-7	43	C	12-3
23	#	8-3	44	D	12-4
24	$	11-8-3	45	E	12-5
25	%	0-8-4	46	F	12-6
26	& ampersand	12	47	G	12-7
27	' apostrophe	8-5	48	H	12-8
28	(	12-8-5	49	I	12-9
29	)	11-8-5	4A	J	11-1
2A	*	11-8-4	4B	K	11-2
2B	+	12-8-6	4C	L	11-3
2C	, comma	0-8-3	4D	M	11-4
2D	–	11	4E	N	11-5
2E	.	12-8-3	4F	O	11-6
2F	/	0-1	50	P	11-7
30	0	0	51	Q	11-8
31	1	1	52	R	11-9
32	2	2	53	S	0-2
33	3	3	54	T	0-3
34	4	4	55	U	0-4
35	5	5	56	V	0-5
36	6	6	57	W	0-6
37	7	7	58	X	0-7
38	8	8	59	Y	0-8
39	9	9	5A	Z	0-9
3A	:	8-2	5B	[opening bracket	12-8-2
3B	;	11-8-6	5C	\ reverse slash	0-8-2
3C	<	12-8-4	5D	] closing bracket	11-8-2
3D	=	8-6	5E	^ circumflex	11-8-7
3E	>	0-8-6	5F	_ underline	0-8-5
3F	?	0-8-7	7B	{ treated as [	
40	@ commercial at	8-4	7D	{ treated as]	

Table A-2. Delimiter Characters

Delimiter	Function	Section Reference
, (comma)	Delimits elements in a statement field.	Section 3 (Statement Structure)
	Delimits elements in a list and arguments in a procedure or function call.	Section 4 (Procedures/functions)
	Delimits subscripts of a set element reference.	Section 4 (Referencing Sets)
( ) parentheses	Enclose arguments of a function call.	Section 4 (Functions)
	Used for grouping in an arithmetic expression or for repetition.	Appendix B (Expressions)
[ ] brackets	Enclose subscripts for referencing a subset of a set; enclose subsets of sets. NOTE The examples in appendix L show the { } characters which are equivalent to [ ]; the programmer must punch [ ]. [: 12-8-2 punch] : 11-8-2 punch	Section 4 (Referencing Sets)
ƀ blank	Terminates a statement field except in a character string constant or comment.	Section 3 (Statement Structure)
" quotes	Encloses character string for a string constant.	Appendix A (Constants)
: colon	Indicates ordinal of an element within a set. Indicates ordinal of a symbol attribute.	Section 4 (Defining Sets)
# pound sign	Indicates start of hexadecimal constant.	Section 4 (RATT)
⌒ circumflex	Used as escape character in a character string constant; indicates the next 2 hex digits form a special ASCII character.	Appendix A (Constants)

Table A-3. Special Characters

Special Character	Function	Section Reference
$	Specifies a drop to a lower level of reference; cannot be used at level 1.	Section 2 (Levels of Symbol Reference)
@	Indicates current value of active location counter. The @ has the same relocation as the active location counter.	Section 4 (Address and Location Control)
*	At beginning of a statement field, indicates the following characters comprise a comment.	Section 3 (Statement Structure)
&	Indicates statement continues at next continuation begin column.	Section 2 (Statement Structure)

CONSTANTS

A constant is a numeric value which cannot be changed by a program. Nine types of constants can be specified in a Control Data STAR assembler program:

Integer	Character String
Integer String	Packed Decimal
Hexadecimal	Zone Decimal
Hexadecimal String	Real
Bit String	

The following paragraphs describe the format which is used when writing each constant type in a program. The rules described here are summarized in table A-4 following the discussion of Real Constants.

INTEGER CONSTANT

An integer constant is a signed string of numeric characters (digits) 0-9. The constant is converted to its signed, 48-bit binary equivalent.

±digit-string

In data generation, the generated length of an integer constant is 64 bits, sign extended to 48 bits.

During data generation, if the integer is truncated, the most significant bits are lost.

```
                                           GEN       #123456789123456789
********** WARNING - CONSTANT TRUNCATED IN OPERAND FIELD
1 000000000240 F    00007891 23456789
```

The maximum significance of the integer is 47 bits excluding sign.

Integer constants are always right justified, sign-extended in data generation.

Maximum integer constant is +140,737,488,355,327; the minimum is -140,737,488,355,328.

Examples:

	Assembler Generated Data	
Integer Constant	**When 64 Bits Requested**	**Default Length Requested**
0	00000000 00000000	00000000 00000000
1	00000000 00000001	00000000 00000001
16	00000000 00000010	00000000 00000010
256	00000000 00000100	00000000 00000100
4096	00000000 00001000	00000000 00001000
65535	00000000 0000FFFF	00000000 0000FFFF
-0	00000000 00000000	00000000 00000000
-1	FFFFFFFF FFFFFFFF	0000FFFF FFFFFFFF
-17	FFFFFFFF FFFFFFEF	0000FFFF FFFFFFEF
-328	FFFFFFFF FFFFFEB8	0000FFFF FFFFFEB8
-55823	FFFFFFFF FFFF25F1	0000FFFF FFFF25F1
-40737	FFFFFFFF FFFF60DF	0000FFFF FFFF60DF

INTEGER STRING CONSTANTS

An integer string constant is written as the letter I followed by a signed string of numeric characters enclosed in quotes. The constant is converted to a signed binary string equivalent.

I"±digit-string"

The integer string constant cannot be used in arithmetic expressions.

In data generation, the default length of an integer string constant is the minimum number of bytes needed to represent the signed binary string.

During data generation, if an integer string is truncated, the most significant bits are lost. When truncation occurs a warning message is generated. "WARNING – CONSTANT TRUNCATED IN OPERAND FIELD."

Integer string constants are right justified, sign-extended in data generation.

Maximum number of digits is 2^{12}.

Examples:

Integer String Constant	Assembler Generated Data: When 64 Bits Requested	Assembler Generated Data: Default Length Required
I"0"	0000000000000000	00
I"1"	0000000000000001	01
I"+16"	0000000000000010	10
I"256"	0000000000000100	0100
I"4096"	0000000000001000	1000
I"+65535"	000000000000FFFF	00FFFF
I"-0"	0000000000000000	00
I"-1"	00000000000000FF	FF
I"-17"	00000000000000EF	EF
I"-328"	000000000000FEB8	FEB8
I"-55823"	0000000000FF25F1	FF25F1
I"-40737"	0000000000FF60DF	FF60DF

HEXADECIMAL CONSTANT

A hexadecimal constant is written as a # (pound sign) followed by a string of hexadecimal characters from the set 0-9 and A-F. The constant is converted to a 48-bit binary equivalent.

±#hexadecimal-character-string

The default length of a hex constant, in data generation, is 64 bits sign extended to 48 bits.

When a hex constant is truncated during data generation, the most significant bits are lost.

```
                                                  GEN    #FFFFFFFFFFFFFFFFF
*********** WARNING - CONSTANT TRUNCATED IN OPERAND FIELD
```

Hexadecimal constants are right justified, sign-extended in data generation.

The maximum hex constant is: ±#FFFF FFFF FFFF

Examples:

Hexadecimal Constant	Assembler Generated Data When 64 Bits Requested	Assembler Generated Data Default Length Requested
#9	0000000000000009	0000000000000009
#F	000000000000000F	000000000000000F
#FE	00000000000000FE	00000000000000FE
#0F	000000000000000F	000000000000000F
+#FF	00000000000000FF	00000000000000FF
#8000	0000000000008000	0000000000008000
#08000	0000000000008000	0000000000008000
-#9	FFFFFFFFFFFFFFF7	0000FFFFFFFFFFF7
-#F	FFFFFFFFFFFFFFF1	0000FFFFFFFFFFF1
-#FE	FFFFFFFFFFFFFF02	0000FFFFFFFFFF02
-#0F	FFFFFFFFFFFFFFF1	0000FFFFFFFFFFF1
-#FF	FFFFFFFFFFFFFF01	0000FFFFFFFFFF01
-#8000	FFFFFFFFFFFF8000	0000FFFFFFFF8000
-#08000	FFFFFFFFFFFF8000	0000FFFFFFFF8000

HEXADECIMAL STRING CONSTANT

A hexadecimal string constant is written as a letter X followed by a string of hexadecimal characters (from the set 0-9 and A-F) enclosed in quotes. Each character in the string is converted to a 4-bit hexadecimal equivalent.

X"hexadecimal-character-string"

The hexadecimal string constant cannot be used in arithmetic expressions.

The default length of a hex string constant, in data generation, is the number of half-bytes (4 bits) required to represent the constant.

Hex string constants are always right justified, zero filled in data generation.

Maximum number of hex digits is 2^{12}.

Examples:

Hexadecimal String Constant	Assembler Generated Data: When 64 bits Requested	Assembler Generated Data: Default Length Requested
X"9"	0000000000000009	9
X"F"	000000000000000F	F
X"FE"	00000000000000FE	FE
X"0F"	000000000000000F	0F
X"FF"	00000000000000FF	FF
X"8000"	0000000000008000	8000
X"08000"	0000000000008000	08000

BIT STRING CONSTANT

A bit string constant is written as a letter B followed by a string of binary digits from the set 0 and 1 enclosed in quotes. Each character in the string is converted to a 1-bit binary equivalent.

B"binary-digit-string"

The bit-string constant cannot be used in arithmetic expressions.

The default length of a bit string constant, in data generation, is the number of bits required to represent the bit string.

Bit string constants are right justified and zero-filled when used in data generation.

Maximum number of bits is 2^{12}.

Examples:

Bit String Constant	Assembler Generated Data When 64 Bits Requested
B"1"	0000000000000001
B"1110"	000000000000000E
B"011000"	0000000000000018
B"0101010101"	0000000000000155
B"1010101010"	00000000000002AA

CHARACTER STRING CONSTANT

A character string constant is written as a string of ASCII characters enclosed in quotes. Each character is converted to an 8-bit byte equivalent representation.

"character-string"

The character string constant cannot be used in arithmetic expressions.

The default length of a character string constant, in data generation, is the number of bytes required to represent the character string.

A circumflex in the character string indicates the next 2 hexadecimal characters are to be combined to form a special ASCII code.

The following characters must be inserted by using the circumflex: "(quote), &(ampersand), and ^ (circumflex), e.g., "^41" = "A"

Character string constants are always left justified and blank-filled in data generation. The data generation field must be a multiple of bytes.

The current control section counter must be byte aligned for data generation of character strings. Automatic alignment occurs if improper alignment is detected. When automatic alignment occurs the message: AUTOMATIC ALIGNMENT PERFORMED FOR DATA TYPE INDICATED LABELS MAY NOT CORRESPOND TO START OF DATA" is issued.

Maximum number of characters is 2^{12}.

Examples:

Character String Constant	Assembler Generated Data: When 192 Bits Requested	Assembler Generated Data: Default Length Requested
"ASSEMBLER"	415353454D424C45 5220202020202020 2020202020202020	415353454D424C45 52
"USES FOR AMPERSAND"	555345532020464F 5220414D50455253 414F442020202020	555345532020464F 5220414D50455253 414F44

PACKED DECIMAL CONSTANT

A packed decimal constant is written as the letter P followed by a signed string of numeric characters enclosed in quotes. The constant is converted to its signed BCD equivalent: the rightmost 4 bits contain the size.

P"±digit-string"

Packed decimal constants cannot be used in arithmetic expressions.

The default length of a packed decimal constant, in data generation, is the number of bytes required to represent the signed packed decimal constant.

The most significant bits are lost when truncation is performed.

Packed decimal constants are always right justified zero-filled in data generation.

Maximum number of digits 2^{12}.

Examples:

Packed Decimal Constants	Assembler Generated Data: When 64 Bits Requested	Assembler Generated Data: Default Length Requested
P"12345"	000000000012345A	12345A
P"+543"	000000000000543A	543A
P"-6789"	000000000006789B	06789B
P"-9876"	000000000009876B	09876B

ZONED DECIMAL CONSTANT

A zoned decimal constant is written as the letter Z followed by a signed string of numeric characters enclosed in quotes. The constant is converted to its signed ASCII-zoned format with the rightmost byte (an overpunched digit) containing the sign and the least significant decimal digit.

Z"±digit-string"

Zoned decimal constants cannot be used in arithmetic expressions.

The default length of a zoned decimal constant, in data generation, is the number of bytes required to represent the signed zoned decimal constant.

The most significant bits are lost when truncation is performed.

The current control section counter must be byte aligned for data generation of zoned constants. Automatic alignment occurs when improper alignment is detected.

Maximum number of digits is 2^{12}.

Examples:

Zoned Decimal Constants	Assembler Generated Data When 64 Bits Requested	Assembler Generated Data Default Length Requested
Z"12345"	3030303132333445	3132333445
Z"+543"	3030303030353443	353443
Z"-6789"	3030303036373852	36373852
Z"-9876	303030303938374F	3938374F

REAL CONSTANT

The formats for signed real constants are:

$\pm n_1.n_2E\pm n_3$ for half word

$\pm n_1.n_2D\pm n_3$ for full word

The real constant is converted to its internal normalized floating-point equivalent.

n_1 is an optional string of numeric characters.

n_2 is a non-empty string of numeric characters.

n_3 is an optional string of numeric characters.

The period is not optional but the E or D and the signs are optional. If neither E nor D is given, the default is E.

When real constants are used in arithmetic expressions, normalized arithmetic is used for add and subtract operations; significant arithmetic is used for multiply and divide operations; and the result is always normalized.

The default length of a real constant, in data generation, is an 8-bit exponent, 24-bit coefficient for E (32-bit value); or a 16-bit exponent, 48-bit coefficient for D (64-bit value).

When a real constant is converted to its internal form, the least significant digits are truncated.

When a real constant is used in data generation, the rightmost bits of the constant are truncated.

Real constants are always right justified, zero-filled in data generation.

For D, maximum number of digits for n_1 and n_2 combined is 14.

For E, maximum number of digits for n_1 and n_2 combined is 7.

For D, maximum number of digits for n_3 is 4.

For E, maximum number of digits for n_3 is 2.

If half- and full-word real constants are mixed in arithmetic expressions, the result is a full word.

Examples:

Real Constants	Assembler Generated Data When 64 Bits Requested
+123.45E+4	FE4B5910
-123.45E+4	FEB4A6F0
+123.45E-4	E3652157
-123.45E-4	E39ADEA9
123.45D+4	FFE64B59 10000000
+123.45D-4	FFCB6521 57689CA0
-123.45D-4	FFCB9ADE A8976360

Table A-4. Summary of Rules for Constants

CONSTANT TYPE/FORMAT	USED IN ARITHMETIC EXPRESSION	TRUNCATION	MAX SIZE/ VALUE	MIN SIZE/ VALUE	DEFAULT LENGTH AT DATA GENERATION	JUSTIFICATION DURING DATA GENERATION	MISCELLANEOUS
INTEGER (+digit string)	YES	most significant bits	+140,737, 488,355, 327	-140,737, 488,355, 328	48 bits sign extended to 64 bits	right justified /sign extended	
INTEGER STRING I"±digit-string")	NO	most significant bits	2^{12} digits		min # of bits required to represent the #.	right justified /sign extended	
HEXADECIMAL (± # hex-char-string")	YES	most significant bits	#FFFF FFFF FFFF		48 bits sign extended to 64 bits	right justified /sign extended	
HEXADECIMAL STRING (X"hex-char-string")	NO	most significant bits	2^{12} hex digits		min # of half-bytes required to represent the #.	right justified zero-filled	
BIT STRING (B"binary-digit-string")	NO	most significant bits	2^{12} bits		# of bits required to represent the string	right justified zero-filled	
CHARACTER STRING ("char-string")	NO	most significant bits	2^{12} characters		# of bytes required to represent the Character String	left justified /blank filled (field generated must be a byte multiple)	Current control section counter must be byte aligned for data generation of character string. This is accomplished atuomatically if programmer fails to ensure byte alignment
PACKED-DECIMAL P"±digit-string")	NO	most significant bits	2^{12} digits		# of bytes required to represent the signed packed Decimal Constant	right justified /zero-filled	

Table A-4. Summary of Rules for Constants (Cont'd)

CONSTANT TYPE/FORMAT	USED IN ARITHMETIC EXPRESSION	TRUNCATION	MAX SIZE/ VALUE	MIN SIZE/ VALUE	DEFAULT LENGTH AT DATA GENERATION	JUSTIFICATION DURING DATA GENERATION	MISCELLANEOUS
ZONED-DECIMAL (Z"±digit-string")	NO	most significant bits	2^{12} digits		number of bytes required to represent the Zoned Decimal Constant	right justified /zero filled; field must be a multiple of bytes	Current control counter must be byte aligned for data generation. automatically accomplished if programmer fails to assure proper alignment.
REAL (±n1.n2E±n3 half word) (±n1.n2D±n3 full word)	YES - normalized add, subtract and normalized significant arithmetic for multiply and divide	1) Internal form least significant bits 2) Data Generation most significant bits.	1) D max # of digits : n1 and n2 (14 digits) n3 (4 digits) 2) E max # of digits: n1 and n2 (7 digits) n3 (2 digits)		32 bit-half word 64 bit full word	right justified /zero filled	When half and full word real constants are mixed in arithmetic operations then result is a fullword value.

SYMBOLS

Symbols are formed by combining 1-63 alphabetic characters or numbers; they provide a convenient means of referring to program elements. Symbols can be used as:

Address identifiers	Form names
Variable identifiers	Procedure names
Function names	Set names
Directive names	

For identifying program elements, all the above symbol types, except directive names and instruction mnemonics, are entered in the label field. The latter two types are entered as described in table A-5. The first character of a symbol must be alpha. The remaining symbols may be numeric or an underscore.

Examples of legal symbols:

A	R_35_X
BAKER	R_1_5
CHARLIE_1	Z_246_8_10

SYMBOL RELATED DIAGNOSTICS

Diagnostics related to the improper construction of a symbol in a label field are listed below.

********* MISSING OPERATOR IN LABEL FIELD

Occurs when a $ or @ is embedded in the symbol, or when a symbol starting with a digit is followed by a letter without an intervening comma.

Examples:

K @ LM	Embedded @
D3$45	Embedded $
1ABCD	Written as 1, A, B, C, D, this would constitute a label list of 5 labels, the first being numeric.

********* UNMATCHED PAREN IN LABEL FIELD

I(123

********* ILLEGAL STRING CONSTANT IN LABEL FIELD

J"BA

********* ILLEGAL SYMBOL IN LABEL FIELD

Occurs when a label field begins with an underscore:

_A123

Table A-5. Symbol Summary

Symbol Type	Location As Identifier	Location As Reference	Comments
Address Identifier	Label field of directives: form call MSEC RES EXT GEN ORG		Value of identifier used in label field is value of P counter after alignment. Relocation attribute is same as that of P counter.
	Label field of program statement.	Command field list/ operand field list of directives or program statement.	Returns value of address identifier when used in command/operand list.
Variable Identifier	Label field of directives RDEF EQU RPT		
		Command field list/ operand field list.	Returns value of identifier when used in command/ operand list.
Function Name	Label field list of NAME directive in a function definition.		
		Any command/operand field list. Function reference format: Function Name (list of operands)	A function reference calls a routine to process function definition statements. When this call is terminated by an EXITP or ENDP directive, the value of the directives operand field list is returned.

Table A-5. Symbol Summary (continued)

Symbol Type	Location As Identifier	Location As Reference	Comments
Directive Name		Command field followed by operands in command list and operand list fields.	This symbol is recognized by assembler. A reference to a directive name is a call to a processor that performs the function of the directive.
Form Name	Label field of FORM directive.		
		Command field followed by operands in operand list.	A form reference is a call to a processor that generates data defined by a form definition and the operands in the form reference.
Procedure Name	Label field of NAME directive in procedure definition.		
		Command field followed by command list and operand list.	A procedure reference is a call to a processor that executes statements in the procedure definition until an EXITP or ENDP directive occurs. No value is returned.
Set Name	Label field of SET directive. Example: $\underbrace{\text{BETA}}_{\text{label}}$ SET 3, 6, 9		
		Command list or operand list fields. Example: GEN $\underbrace{\text{.ELM.BETA}}_{\text{operand}}$	Returns a value of complete set list, contained in brackets.

Table A-5. Symbol Summary (continued)

Symbol Type	Location As Identifier	Location As Reference	Comments
Instruction Mnemonic	——	Command field followed by command and operand lists. Symbol recognized by the Control Data STAR system as a machine instruction mnemonic.	Calls processor that generates the machine instruction as data.
Numeric Label	1 to 14 numeric characters (leading zeros preceding the label field list are ignored.	Operand field of RPT a and GOTO directives.	Example: reference GOTO 5 . . . 5 - - identifier in label field

EXPRESSIONS B

The Control Data STAR assembler permits the use of simple expressions, consisting of one symbol, and complex expressions, consisting of two or more symbols connected by an operator. For expressions with more than one operation, the order in which each operation is evaluated is determined by the hierarchial level assigned to the operators.

Expressions may be arithmetic, relational, logical, or special. Table B-1 lists the operators for each expression type, and includes interpretation of each operator, as well as the hierarchial value assigned to it. After reading this appendix, refer to figure B-1 which illustrates the evaluation of a logical expression.

Set or function names cannot be used as an operand in an expression; however, function call with parameter lists can.

Unary operators must preceed an operand

A unary operator can follow a binary operator without parentheses.

.BS+4 (valid). Binary operator

.NOT.-A (invalid unary followed by another unary operator). Must be .NOT.(-A)

Table B-1. Operators

Type	Operator	Interpretation	Heirarchy
Arithmetic	+	Unary plus	1
	+	Addition	4
	-	Unary minus	1
	-	Subtraction	4
	.BS. (binary scale)	Shift operands to the left of the operator at assembly time (+ or missing shift left; - shift right) by the number of bit positions specified by the value to the right of the operator. e.g., A.BS.+4	2
	*	Multiplication	3
	/	Division	3
	.GE.	Condition true if greater than or equal to	5
Comparison	.EQ.	Condition true if equal	5
	.NE.	Condition true if not equal	5
	.GT.	Condition true if greater than	5
	.LT.	Condition true if less than	5
	.LE.	Condition true if less than or equal to	5
Logical	.NOT.	Logical one's complement (unary)	1
	.AND.	Logical product	7
	.OR.	Logical or (inclusive or)	8
Special	.CAT.	Concatenate character string on the left to that on the right of this operator. Operands can be: expressions, character string, function designator, variable identifier, or set designator. All types must evaluate to a character string prior to concatenation. Result must be a character string. e.g., "STAR" .CAT. " ASSEMBLER" results in STAR ASSEMBLER.	1
	.ELM.	Expand a set to a list of elements.	1
	.NR. (ignore relocation)	Convert the address (external or relocatable) to a 48 bit integer constant by removing the relocation ordinal. This occurs at assembly time.	
	: (positional operator)	Give operand to the right the list position specified by the operand to the left.	1
	N( )	Repetition operator for a list of (elements) where N is an expression representing a repetition count. N must evaluate to an integer and the elements to be repeated can be of any operand type permitted in as assembler expression including a null.	1

EXPRESSION EVALUATION

Expressions are evaluated left to right, the operations with lower numbered hierarchies are performed first. Parenthesized sub-expressions are expanded from the inside and are performed first. Operators of equal hierarchy are evaluated left to right.

Operations involving the use of relocatable address cannot be performed in the code section of the subprogram; i.e., must be performed in the data section. If an operation involving the use of a relocatable address is attempted in a code section the following message is generated.

    ********** RELOCATION NOT PERMITTED IN CODE MSEC

ARITHMETIC OPERATIONS

Arithmetic operators can generate either an integer constant (which could have been associated with a memory section ordinal) or a real constant. Integer constants and real constants cannot be mixed in an operation. Tables B-3 through B-6 list legal combinations of operand types used in arithmetic operations.

RELATIONAL OPERATIONS

The result of a relational operation is an integer constant zero if the operation proves false, or an integer constant one if the operation proves true. The comparison method for all relational operations is specified in table B-2; a description of allowable combinations of operand types in relational expressions appear in table B-7.

Table B-2. Comparison Methods

Operand Types	Method
Character, bit, and hexadecimal string constant comparison	Bit comparison. When lengths differ, they are considered not equal.
Real constant comparison	Floating-point compare
Packed and zoned decimal constant comparisons	Decimal compare
Integer and hex constant comparison	Signed integer compare
Integer-string constant comparison	Binary compare

EXPRESSION MODE AND EVALUATION

As performed by the assembler, expression evaluation determines the data types of the operands and the specification of a result and data type based on predefined rules. A mode value, assigned by the assembler, describes each data type (operand) used in an expression:

Mode Value	Meaning
0	Not a value; for example, set-of-function name
1	Absolute address
2	Relocatable address
3	External address
4	Integer or hexadecimal constant
5	Hexadecimal string constant
6	Bit string constant
7	Character string constant
8	Real constant
9	Packed decimal constant
10	Zoned decimal constant
11	Integer string constant
12	Null element; element of set list is not defined. Element value is zero.

The following tables (B-3 through B-6) provide the allowable combinations of operand types (modes) for a given operation and the data type (mode) of the result of the operation. The mode result of each operation is contained within the appropriate blocks. An asterisk result indicates that the combination of operands is not permitted.

Table B-3. Unary + - Operations

UNARY + -	Right Operand				
	Relocatable Address	**Integer Constant**	**Hex Constant**	**Real Constant**	**Absolute Address**
	Relocatable Address	Integer Constant	Hex Constant	Real Constant	Absolute Address

Table B-4. Binary Scale Operations (.BS.)

Left Operand \ Right Operand	Integer Constant	Hex Constant	Real Constant
Integer	Integer Constant	Integer Constant	*
Hex Constant	Hex Constant	Hex Constant	*
Real Constant	Real Constant	Real Constant	*

For example:

```
                  00 000000000003     C      EQU      3
01 000000000040 E 00000000 00000006          GEN       C.BS.+1
01 000000000080 F 00000000 00000001          GEN        C.BS.-1

                 00 0000000000006    AA     EQU   #3.BS.+#1
                 00 0000000000006    BB     EQU   #3.BS.+1
                 00 00000000000B2    CC     EQU   89.BS.+#1
```

Table B-5. Multiply and Divide Operations (* /)

Left Operand \ Right Operand	Integer Constant	Hex Constant	Real Constant
Integer Constant	Integer Constant	Integer Constant	*
Hex Constant	Hex Constant	Hex Constant	*
Real Constant	*	*	Real Constant

Table B-6. Add and Subtract Operations (+ -)

		Right Operand				
		External Address	**Relocatable Address**	**Integer Constant**	**Hex Constant**	**Real Constant**
Left Operand	External Address	External Address	*	External Address	External Address	*
	Relocatable Address	*	Relocatable Address	Relocatable Address	Relocatable Address	*
	Integer Constant	*	Relocatable Address	Integer Constant	Integer Constant	*
	Hex Constant	External Address	Relocatable Address	Hex Constant	Hex Constant	*
	Real Constant	*	*	*	*	Real Constant

Table B-7. Relational Operations (EQ, NE, GT, GE, LT, LE)

Left Operand \ Right Operand	Relocatable Address	Integer Constant	Hex Constant	Hex-String Constant	Bit-String Constant	Char-String Constant	Real Constant	Packed-Decimal Constant	Zoned-Decimal Constant	Integer-String Constant
Relocatable Address	INT Constant	*	*	*	*	*	*	*	*	*
Integer(INT) Constant	*	INT Constant	INT Constant	*	*	*	*	*	*	*
Hex Constant	*	INT Constant	INT Constant	*	*	*	*	*	*	*
Hex-String (STR) Constant	*	*	*	INT Constant	*	*	*	*	*	*
Bit-String Constant	*	*	*	*	INT Constant	*	*	*	*	*
Char-String Constant	*	*	*	*	*	INT Constant	*	*	*	*
Real Constant	*	*	*	*	*	*	INT Constant	*	*	*
Packed-Decimal Constant	*	*	*	*	*	*	*	INT Constant	*	*
Zoned-Decimal Constant	*	*	*	*	*	*	*	*	INT Constant	*
Integer(INT) String(STR) Constant	*	*	*	*	*	*	*	*	*	INT Constant

LOGICAL OPERATIONS

Logical operations are performed left to right and bit by bit. If operands are unequal in length, the shorter is left justified and right extended with zeros until both are equal in length. Allowable combinations of operands in logical operations appear in table B-8.

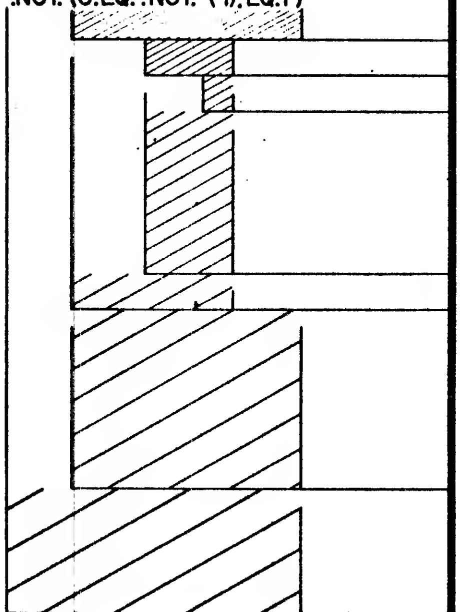
Table B-8. Logical Operations (AND, OR)

For a unary .NOT. operation, the result length is that of the operand being evaluated.

<table>
<tr><th></th><th></th><th colspan="8">Right Operand</th></tr>
<tr><th></th><th></th><th>Integer Constant</th><th>Hex Constant</th><th>Integer-String Constant</th><th>Bit-String Constant</th><th>Char-String Constant</th><th>Real Constant</th><th>Packed-Decimal Constant</th><th>Zoned-Decimal Constant</th></tr>
<tr><td rowspan="9">Left Operand</td><td>Relocatable Address</td><td></td><td></td><td colspan="4">Mode and length of left operand</td><td></td><td></td></tr>
<tr><td>Integer Constant</td><td></td><td></td><td colspan="4">Mode and length of left operand</td><td></td><td></td></tr>
<tr><td>Integer-String Constant</td><td></td><td></td><td colspan="4">Mode and length of left operand</td><td></td><td></td></tr>
<tr><td>Hex Constant</td><td></td><td></td><td colspan="4">Mode and length of left operand</td><td></td><td></td></tr>
<tr><td>Bit-String Constant</td><td></td><td></td><td colspan="4">Mode and length of left operand</td><td></td><td></td></tr>
<tr><td>Char-String Constant</td><td></td><td></td><td colspan="4">Mode and length of left operand</td><td></td><td></td></tr>
<tr><td>Real Constant</td><td></td><td></td><td colspan="4">Mode and length of left operand</td><td></td><td></td></tr>
<tr><td>Packed-Decimal Constant</td><td></td><td></td><td colspan="4">Mode and length of left operand</td><td></td><td></td></tr>
<tr><td>Zoned-Decimal Constant</td><td></td><td></td><td colspan="4">Mode and length of left operand</td><td></td><td></td></tr>
</table>

EXPRESSION

.NOT. (O.EQ. .NOT. (-1).EQ.1)



EXPLANATION	HEX REPRESENTATION	MODE	OPERATOR	HIERARCHY
a) PARENS EVALUATED FIRST			NOT	1
1) NOT HAS HIGHEST HIERARCHY			EQ	5
a) PARENS EVALUATED				
1) − IS A UNARY OPERATOR AND INDICATES 2'S COMPLEMENT			−	1
2) 1 IS A 48-BIT INTEGER CONSTANT	0000 0000 0001	INTEGER CONSTANT		
3) THE COMBINATION OF THE TWO	FFFF FFFF FFFF	INTEGER CONSTANT		
b) NOT PERFORMS A 1'S COMPLEMENT	0000 0000 0000	INTEGER CONSTANT		
2) SINCE THERE ARE 2 EQ'S, EVALUATION CONTINUES FROM LEFT TO RIGHT.			EQ	5
a) 0 IS A 48-BIT INTEGER CONSTANT	0000 0000 0000	INTEGER CONSTANT		
b) WHEN RELATIONAL OPERATORS ARE EVALUATED THEY RETURN A 48-BIT INTEGER CONSTANT OF ONE FOR TRUE OR ZERO FOR FALSE.				
c) THE EXPRESSION EVALUATES TO BE TRUE	0000 0000 0001	INTEGER CONSTANT		
3) COMPLETE EVALUATION OF PARENS			EQ	5
a) 1 IS A 48-BIT INTEGER CONSTANT	0000 0000 0001	INTEGER CONSTANT		
b) THE EXPRESSION EVALUATES TO BE TRUE	0000 0000 0001	INTEGER CONSTANT		
b) THE COMPLETE EXPRESSION IS NOW NOTTED [1'S COMPLEMENT]	FFFF FFFF FFFF	INTEGER CONSTANT		

Figure B-1. Expression Hierarchial Evaluation

STAR MACHINE INSTRUCTIONS C

STAR instructions may be classified into ten categories: Register, Index, Branch, Vector, Sparse Vector, Vector Macro, String, Logical String, Non-Typical, and Monitor. Instruction size is either 32 bits or 64 bits and formats vary within an instruction group.

GENERAL FORMAT

The general format for a symbolic machine instruction is identical to that of a procedure reference:

Numeric Label, List		Mnemonic, Qualifiers		Operands

LABEL FIELD

The label field consists of an optional numeric label followed by an optional list of symbols separated by commas. The symbols are defined to be address identifiers and are given the value of the current location counter after alignment. They are used to define locations at assembly time and do not become part of the 32-bit or 64-bit instructions.

COMMAND FIELD

The command field consists of mnemonics and associated qualifiers. Mnemonics specify the machine instruction to be generated. (They are mapped into the 8-bit function field.) Every instruction function code has a different mnemonic. The mnemonic symbol can be used as an address identifier, variable identifier, set name, and function name without redefining the mnemonic as a machine instruction. Defining a mnemonic symbol to be a procedure name or form name results in instruction redefinition; therefore, use of that machine instruction is lost.

Command field qualifiers are lists of symbols that indicate a sub-operation of the function code specified by the instruction mnemonic. Qualifiers are not reserved symbols and definition of a qualifier symbol by a user does not alter its value as qualifier to an instruction. The user can define his own qualifiers, provided the symbols differ from those qualifiers supplied by the assembler. The assembler checks user defined qualifiers to ensure that the sub-operation specified can be performed. Assembler supplied qualifiers are listed in table C-1.

Table C-1 Qualifiers

Qualifier	Meaning	Hex Value	Default (value is 00)
A	Broadcast A operand	10	No broadcast of A
B	Broadcast B operand	08	No broadcast of B
BR	Branch unconditionally	40	Do not branch
BRB	Branch backward	06	Branch to (Y) + (B)
BRF	Branch forward	04	Branch to (Y) + (B)
BRO	Branch on one	80	Do not branch
BRZ	Branch on zero	C0	Do not branch
C	Complement A operand	02	Normal A operand
CH	Destination C is half word	08	Destination C is full word
D	Character delimiter for A and B	80	Count delimited for A and B operands
DC	Character delimiter for destination C	20	Count delimited for destination C
DD	Double character delimiter for A and B operands	C0	Count delimited for A and B operands
DDC	Double character delimiter for destination C	30	Count delimited for destination C
DM	Character mask delimiter for A and B operands	40	Count delimited for A and B operands
H	Half word operand	80	Full word operands
LH	Start at last hit	20	Starts over
MA	Magnitude of A operand	04	Normal A operand
MB	Magnitude of B operand	01	Normal B operand
N	Negative A operand	06	Normal A operand
NCC	No conflict checking	01	Conflict checking
NIX	Do not increment X	04	Increment
NIY	Do not increment Y	02	Increment
NIZ	Do not increment Z	01	Increment
NS	Packed to zoned no sign	C0	Normal zone sign
O	Offset destination and control vector	20	No offset
SO	Set bit to one	20	Do not alter bit
SS	Zoned 8 bit sign to packed or packed to zoned 8 bit sign	80	Normal zone sign
SZ	Set bit to zero	30	Do not alter bit
T	Toggle bit	10	Do not alter bit
Z	Control vector on zeroes	40	Control vector on ones

OPERAND FIELD

The instruction operand field lists all operands to be used with the instruction. Combination of operand types that can be used with an instruction depends on the format type for the instruction. Twelve format types (categories) are available. A particular form type is usually, but not necessarily, common to a group or groups of instructions.

Operand Form	Meaning
[OP1,OP2]	Operand 1 offset or indexed by operand 2 (see table C-10 vector instructions)
[OP1]	Operand 1 offset or indexed by zero
[,OP2]	Zero offset or indexed by operand 2

Each format type includes a corresponding instruction designator portion. Most formats are divided into lengths of 8-bit characters. The following drawings illustrate available instruction formats and specify the contents of each format division. Cross-hatching denotes undefined areas which must be zero filled. The assembler automatically generates zero fill for these areas. A description of the designators used in the format layouts appears in table C-2.

0 – 7	8 – 15	16 – 23	24 – 31	32 – 39	40 – 47	48 – 55	56 – 63
F (FUNCTION)	G (SUB-FUNCTION)	X (OFFSET FOR A)	A (LENGTH AND BASE ADDRESS)	Y (OFFSET FOR B)	B (LENGTH AND BASE ADDRESS)	Z (CONTROL VECTOR BASE ADDRESS)	C (LENGTH AND BASE ADDRESS)
							C + 1 (OFFSET FOR C & Z)

FORMAT 1 - USED FOR VECTOR, VECTOR MACRO, AND SOME NON-TYPICAL INSTRUCTIONS

0 – 7	8 – 15	16 – 23	24 – 31	32 – 39	40 – 47	48 – 55	56 – 63
F FUNCTION	G (SUB-FUNCTION)	X (ORDER VECTOR LENGTH & BASE)	A (BASE ADDRESS)	Y (ORDER VECTOR LENGTH AND BASE ADDRESS)	B (BASE ADDRESS)	Z (ORDER VECTOR LENGTH AND BASE ADDRESS)	C RESULT LENGTH AND BASE ADDRESS

FORMAT 2 - USED FOR SPARSE VECTOR AND SOME NON-TYPICAL INSTRUCTIONS

0 – 7	8 – 15	16 – 23	24 – 31	32 – 39	40 – 47	48 – 55	56 – 63
F (FUNCTION)	G (SUB-FUNCTION)	X (INDEX FOR A)	A (LENGTH AND BASE ADDRESS)	Y (INDEX FOR B)	B (LENGTH AND BASE ADDRESS)	Z (INDEX FOR C)	C (LENGTH AND BASE ADDRESS)

FORMAT 3 - USED FOR LOGICAL STRING AND STRING INSTRUCTIONS

0 – 7	8 – 15	16 – 23	24 – 31
F (FUNCTION)	R (SOURCE 1)	S (SOURCE 2)	T (DESTINATION)

FORMAT 4 - USED FOR SOME REGISTER, ALL MONITOR, THE 3D AND 04 NON-TYPICAL INSTRUCTIONS

0–7	8–15	16–63
F (FUNCTION)	R (DESTINATION)	I (48 BITS)

FORMAT 5 - USED FOR THE BE, BF, CD AND CE INDEX INSTRUCTIONS AND FOR THE B6 BRANCH INSTRUCTION

0–7	8–15	16–31
F (FUNCTION)	R (DESTINATION)	I (16 BITS)

FORMAT 6 - USED FOR THE 3E, 3F, 4D AND 4E INDEX INSTRUCTIONS AND THE 2A REGISTER INSTRUCTION

0–7	8–15	16–23	24–31
F (FUNCTION)	R	S	T (BASE ADDRESS)

FORMAT 7 - USED FOR SOME BRANCH AND NON-TYPICAL INSTRUCTIONS

0–7	8–15	16–23	24–31
F (FUNCTION)	R (REGISTER)	S (REGISTER)	T (BASE ADDRESS)

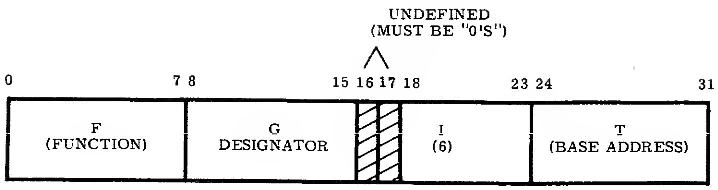
FORMAT 8 - USED FOR SOME BRANCH INSTRUCTIONS

0–7	8–15	16–23	24–31
F (FUNCTION)	G DESIGNATOR	S (BIT TEST ADDRESS)	T

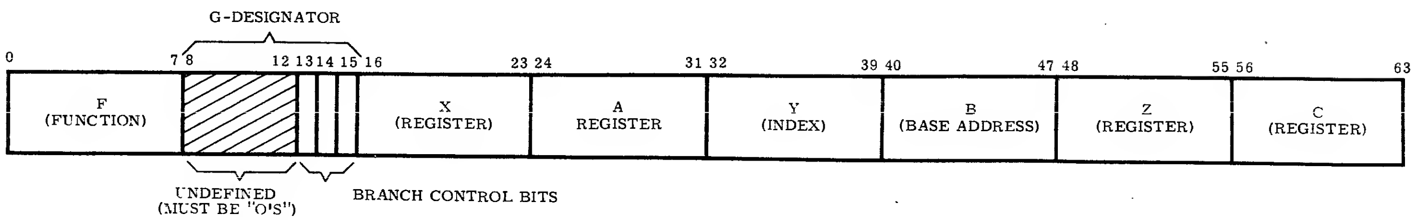
FORMAT 9 - USED FOR THE 32 BRANCH INSTRUCTION

0–7	8–15	16–23	24–31
F (FUNCTION)	R (OLD STATE)	UNDEFINED (MUST BE "0'S")	T (NEW STATE)

FORMAT A - USED FOR SOME INDEX, BRANCH, AND REGISTER INSTRUCTIONS



FORMAT B - USED FOR THE 33 BRANCH INSTRUCTION



FORMAT C - USED FOR THE B0-B5 BRANCH INSTRUCTIONS

Table C-2. Instruction Designators

Designator	Format Type	Definition
A	1 & 3	Specifies a register that contains a field length and base address for the corresponding source vector or string field.
	2	Specifies a register that contains the base address for a source sparse vector field.
	C	Specifies a register that contains a two's complement integer in the rightmost 48 bits.
B	1 & 3	Specifies a register that contains a field length and base address for the corresponding source vector or string field.
	2	Specifies a register that contains the base address for a source sparse vector field.
	C	Specifies a register that contains the branch base address in the rightmost 48 bits.
C	1, 2, & 3	Specifies a register that contains the field length and base address for storing the result vector, sparse vector, or string field.
	C	Specifies the register that will contain the two's complement sum of (A) + (X) in the rightmost 48 bits. The leftmost 16 bits are cleared.
C + 1	1	Specifies a register containing the offset for C and Z vector fields.
d	9 & B	2-bit designator specifying branch conditions.
e	9 & B	2-bit designator specifying object bit altering conditions for the corresponding branch instructions.
F	1 - C	8-bit designator used in all instruction format types to specify instruction function code. It is always contained in the leftmost 8 bits of the instruction and is expressed in hexadecimal for all instruction descriptions. Thus, the function code range is 00-FF_{16}; however, not all possible function codes are used.
G	1, 2, 3, 9, B, & C	8-bit designator specifies certain sub-function conditions. Sub-functions include length of operands (32- or 64-bit), normal or broadcast source vectors, etc. The number of bits used in the G designator varies with instructions.

Table C-2. Instruction Designators (Cont'd)

Designator	Format Type	Definition
I	5	48-bit index used to form the branch address in a B6 branch instruction. In BE and BF index instructions, I is a 48-bit operand.
	6	In 3E and 3F index instructions, I is a 16-bit operand.
	B	In the 33 branch instruction, the 6-bit I is the number of the DFB object bits used in the branching operation.
R	4	In the register and 3D instructions, R is the register containing an operand to be used in an arithmetic operation.
	5 & 6	In the 3E, 3F, BE, and BF index instructions, R is a destination register for the transfer of an operand or operand sum. In the B6 branch instruction, this register contains an item count used to form the branch address.
	7, 8, & A	R specifies registers and branching conditions given in the individual instruction descriptions
S	4	In the register and 3D instructions, S is a register containing an operand to be used in an arithmetic operation.
	7, 8, & 9	S specifies registers and branching conditions given in the individual instruction descriptions
T	4	T specifies a destination register for the transfer of the arithmetic results.
	7, 8, 9, & B	T specifies a register that contains the base address and, in some cases, the field length of the corresponding result field or branch address.
	A	T specifies a register containing the old state of a register, DFB register, etc; in an index, branch, or inter-register transfer operation.
X	1 & 3	Specifies a register that contains the offset or index for vector or string source field A.
	2	Specifies a register that contains length and base address for order vector corresponding to source sparse vector field A.
	C	In the B0-B5 Branch instructions; this register contains a signed, two's-complement integer in the rightmost 48 bits used as an operand in the branching operation

Table C-2. Instruction Designators (Cont'd)

Designator	Format Type	Definition
Y	1 & 3	Specifies a register that contains the offset or index for vector or string field B.
	2	Specifies a register that contains the length and base address for the order vector corresponding to source sparse vector field B.
	C	In the B0-B5 Branch instructions, Y specifies a register that contains an index used to form the branch address.
Z	1	Z specifies a register that contains the base address for the order vector used to control the result vector in field C.
	2	Z specifies a register that contains the length and base address for the order vector corresponding to result sparse vector field C.
	3	Z specifies a register that contains the index for result field C.
	C	In the B0-B5 Branch instructions, Z specifies a register that contains a signed, two's-complement integer in the rightmost 48-bits. It is used as the comparison operand in determining whether the branch condition is met.

INSTRUCTION TYPES

Each STAR instruction type is discussed in the following paragraphs. Tables C-6 through C-15 list the instructions including: OP code, format (F) instruction mnemonic, applicable operand types, qualifiers, and concise description.

The following categories are described:

Register	Vector Macro
Index	String
Branch	Logical String
Vector	Non-Typical
Sparse Vector	Monitor

For a complete description of each instruction included in the STAR set, see Engineering Specification 11845800 (STAR INSTRUCTION DESCRIPTIONS).

REGISTER INSTRUCTIONS

The STAR register file consists of 32- and 64-bit registers. To accommodate the use of both register types, the STAR instruction set includes instructions which access the register file as half words (32 bits) or full words (64 bits).

F	R	S	T

In the register instructions, all source and result destinations are registers; R, S, T, each designate the contents of one of 256 registers. Unless specified, in register-to-register operations the source registers are unchanged and the destination registers are cleared before the result is entered.

Any register except 00_{16} can contain one or both source operands or a result. For a description of the proper use of register 00_{16}, see the Chapter 3, Register File description (paragraph 3.1.7), in Engineering Specification 11845800 (STAR INSTRUCTION DESCRIPTIONS).

INDEX INSTRUCTIONS

Index instructions are used primarily for numerical calculations on field lengths and addresses. The index instructions manipulate either the low order 24 bits of a half word or the low order 48 bits of a full word in designated operational registers. Some index instructions are used for manipulating the high order 8 bits of a half word or the high order 16 bits of a full word in the designated operational registers.

BRANCH INSTRUCTIONS

The branch instructions can be used to compare or examine single bits, 48-bit indexes, 32-bit floating-point operands, or 64-bit operands. Results of comparison determine whether the program continues with the next sequential instruction (branch condition not met) or branches to a different instruction sequence (branch condition met). The instruction sequence can consist of one or more instructions beginning at the branch address specified in the branch instruction format. For instructions which require index operations, all item counts are in half-word increments.

The following comparison rules apply to branch instructions.

If the signs of the coefficients of two operands are unlike, the operands are unequal.

If one operand is indefinite, the compare condition is not met since indefinite is not $>$ $<$ or $=$ to any other operand. If both operands are indefinite the $=$ and $\geqslant$ conditions can be met since indefinite equals indefinite.

If neither operand is indefinite but both operands are machine zero:

A non-indefinite, machine-zero operand with a positive, non-zero coefficient is greater than machine zero.

A non-indefinite, non-machine zero operand with a negative coefficient is less than machine zero.

Machine zero is considered equal only to itself and to any number having a finite exponent and a zero coefficient.

Machine zero is represented as:

8X XXXXXX (32 bits)

8XXX XXXXXX XXXXXX (64 bits)

where: X equals any hexadecimal digit.

An indefinite number is represented as:

7X XXXXXX (32 bits)

or

7XXX XXXXXX (64 bits)

where: X equals any hexadecimal digit.

VECTOR INSTRUCTIONS

The vector instructions perform operations on ordered elements (scalars). These instructions read the scalars, in 32-bit or 64-bit floating-point operand form, from consecutive storage locations over a specified address range (field). Vector instructions perform a designated operation on each set of operands and store the results in consecutive addresses of a result field, beginning with a specified address. A vector can contain as many as 65, 536 items.

The following terms are critical to the understanding of the vector instructions, these terms are fully described in Engineering Specification 11845800.

Order Vector (OV) – A bit string denoting non-significant elements in a vector field. An order vector can be generated by compare instructions and used by compress instructions to generate a sparse vector. The number of ones in the order vector determines field length of sparse vector operands. A filled result (order vector) terminates sparse vector instructions.

Sparse Vector (SV) – Vector field contracted by removing the non-significant elements to conserve storage space and calculation time. Positional significance of the elements is retained by an order vector for each sparse vector.

Control Vector (CV) – Base address of control vector is contained in Z field of vector instructions and vector macro instructions. A control vector determines how many results (C elements) are stored during execution of vector instructions and determines which pairs of A and B elements are compared during Vector Macro operations. Use is specified in an instruction by Z-designator $\neq 0$; the Z designator becomes the CV base address.

Broadcast – Repeated transmission of the same vector element from the register file. Selection of a broadcast or normal element is specified by the state of the G designator of the applicable vector instruction. (See Qualifiers)

Offset – Number used to modify the base address of operands in vector and some non-typical instructions. An offset can be in half words or words (determined by number of bits in operand up to ± 2^{15}-1).

Significance – Bit count for a floating point number which is equal to the number of bit positions in the coefficient (excluding the sign bit) minus the left shift count required to normalize the number.

Control vector, offset, as well as, operand sign content and size are selected through sub-function bits in the vector instruction. These sub-functions are listed in table (C-3).

If the Z designator in format 1 instruction is zero, a control vector is not used; therefore bit 9 becomes undefined. If bits 11 and/or 12 of G = 1, the A and/or B designators denote a constant used as each element of the respective vector field. The instruction ignores associated offsets in this case. The registers specified by A and/or B contain these constants.

Table C-3. Vector Instruction Sub-function Bits

Bit	State	Sub-function
8	0	64-bit operands (words)
	1	32-bit operands (1/2 words)
9	0	Control vector operates on 1's
	1	Control vector operates on 0's
10	0	No offset for result field and control vector
	1	Offset for result field and control vector
11	1	Normal source vectors – A
	1	Broadcast repeated (A)
12	0	Normal source vectors – B
	1	Broadcast repeated (B)
13 14 15	X X X	Sign control† (These bits must be 0 for all instructions other than 80, 81, 82, 84, 85, 86, 88, 89, 93††, 8B, 8C, 8F, CF, D8††, and D9†† instructions. See table C-4.

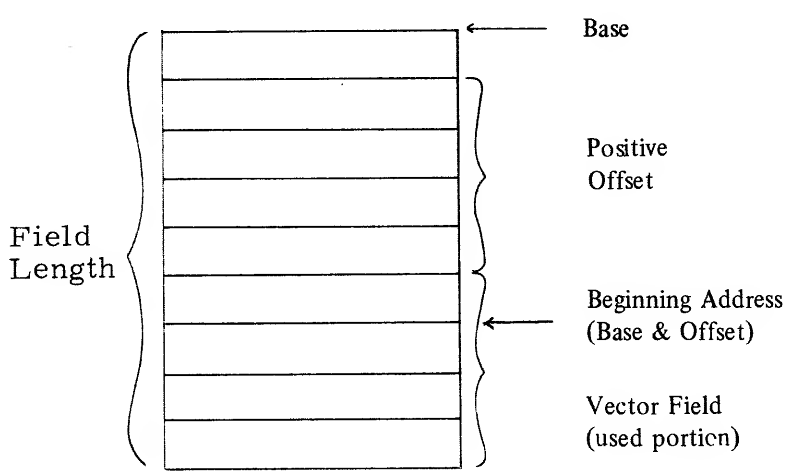
†If both vectors A and B are broadcast constants, instructions that do not terminate by filling the result field (e.g., Select instructions C0 -C3) produce undefined results.
††In these instructions, only bits 13 and 14 are used. Bit 15 must be 0.

Table C-4. Vector Instruction Sign Control Sub-function Bits

Bit 13	Bit 14	Bit 15	Control Operation
0	0	0 or 1	Operands from the A stream are used in normal manner.
0	1	0 or 1	Coefficients of operands from the A stream are 2's complemented before they are used. Any required significance calculation is performed before complementing.
1	0	0 or 1	Magnitude of operands from the A stream is used.
1	1	0 or 1	Coefficients of all positive operands from the A stream are made negative before they are used. Negative operands are not altered.
0 or 1	0 or 1	0	Operands from the B stream are used in normal manner.
0 or 1	0 or 1	1	Magnitude of coefficients of operands from the B stream is used.

Field lengths, Base Address, and Offsets

The operation of subtracting the offset from the field length must result in a positive vector length less than 2^{16} in magnitude. If the resulting vector does not meet these requirements, it is treated as a zero vector length. The beginning address is obtained by adding the offset (including sign extension) to the base address.



CONTROL VECTOR

When the format 1 instruction specifies a control vector (Z designator = 0), a single bit from the vector controls how each element is stored in the result field. When a bit from the control vector prohibits the storing of a result element, the instruction does not alter the previous contents of the corresponding storage address. Therefore, the nth bit read from the control vector prohibits or permits the storing of the nth result in the result vector field.

As specified in Table C-3, bit 9 of the G designator selects whether a 0 or 1 control vector bit permits the result to be stored. If bit 9 of the G designator is a 0 or a 1, the instruction stores the nth result provided the nth bit of the control vector is identical to that specified in the G designator.

The rightmost 48 bits of the register designated by Z contain the base address of the control vector. The control vector field length is the same as the field length for result vector C.

The addition of the offset and base address provides the starting address of the control vector. Since offsets are item counts, the result vector and control vector use the same offset; however, the control vector offset represents a bit offset.

VECTOR INSTRUCTION TERMINATION

Vector instructions terminate when the result vector field is filled. In format 1, when the C designator is zero or the modified field length is zero or negative, the instruction becomes a no-operation (no-op) instruction. The modified C vector length equals the C vector length minus the offset. If the instruction uses no C vector offset, the modified field length equals the C vector field length. The instruction extends short or zero length source vector fields, as required, with machine zeros in additive operations or normalized source vector fields in multiply or divide operations.

VECTOR MACRO INSTRUCTIONS

Vector macro instructions perform operations similar to vector instructions; however, some vector macro instructions do not form result vector fields. For these instructions, the control vector contains neither length nor offset; rather it controls the use of source vector elements.

Bit 10 of the G designator for this instruction must be set to 0. Designators C and C + 1 denote 32 bits when bit 8 of the G designator specifies 32 bit operands.

The control vector for macro instructions which produce result vector fields, performs the same function as in a vector instruction. Vector macro instructions with result field(s), extend short source fields with zeros; they become no-operations, and terminate in an identical manner as a vector instruction. Vector macros with result field(s) terminate when either source vector is exhausted; they do not zero extend short source fields.

Broadcasting both source fields for vector instructions with a result field, produces an undefined condition.

SPARSE VECTOR INSTRUCTIONS

Arithmetic operations can reduce the number elements of a vector field to zero or near-zero value; therefore, except for positional significance, they need not be carried along as floating-point numbers. To conserve both storage and calculation time, a group of sparse vector instructions which permit the expansion and compression of vectors can be used. Similarly, the programmer may wish to eliminate out-of-range data.

The user can form a sparse vector by generating an order vector through the compare instructions. A vector containing non-significant elements can be reduced then to a sparse vector through the (BC) compress instruction which uses the generated order vector to remove the non-significant elements. The operation codes for the compare and compress instructions are C1-C7. The sparse vector can be restored back to the original vector size through MASKV instruction (operation code BB). The format of the sparse vector cannot be distinguished from that of any other vector; however, the associated order vector determines the positional significance of each vector element. Bits, 5, 6, and 7 of the G field must be set to 0, for all sparse vector instructions except those with operation codes: A0-A2, A4-A6, A8, A9, AB, AC, and AF. The paragraph on sign control at the end of this appendix explains bits 5, 6, and 7. When these bits are set to a value, all the G field bits must be zero.

Neither indexing nor offsetting is performed by the sparse vector instructions. The field lengths associated with source sparse vectors A and B are not used (format 2). These lengths are determined by the number of ones in the associated order vector. The field lengths of source order vectors X and Y and the result order vector Z (format 2) are item counts in bits.

SPARSE VECTOR ADD

This example (12) illustrates a method of producing sparse vectors and the use of the add sparse vector instruction. In a sparse vector, extraneous information has been removed; but, the position of its elements remain the same through use of an order vector. This example illustrates:

How to reduce a MATRIX to a sparse vector

How to create an order vector

How to write a sparse vector instruction.

This example also makes use of a broadcast constant.

CREATING THE MATRIX

Matrices are created in this example through GEN directives. Since the MATRIX is a group of vectors, it must have a descriptor specifying its length and base address; and since the instructions using these descriptors require them to be in a register, each descriptor must be equated to a register. Matrices for this example follows:

	Matrix A								**Matrix B**							
Row 1	1	2	3	4	5	6	7	8	25	11	25	10	23	22	21	20
2	9	10	11	12	13	14	15	16	19	18	17	12	15	14	13	12
3	17	18	19	20	21	22	23	24	13	14	15	16	17	18	19	20

Matrix C contains only one element, which is broadcast to create the order vector. The order vectors are created by the CMPGE instructions. These instructions compare the broadcast constant against each item in matrix A and B.. Since the value C is in hexadecimal and the values generated for the matrix were decimal for all integers greater than or equal to 12, a 1 will be placed in all corresponding order vector location. For values less than 12, a zero will be entered in the order vector..

Order Vector for Matrix A

0	24	64
0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1		

Order Vector for Matrix B

0	24	64
1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1		

The matrix contains full-word values; the order vector contains bit values.

Now that an order vector is established, the compress (CPSV) vector instruction can be used to create the sparse vector.

Compressed Matrix A

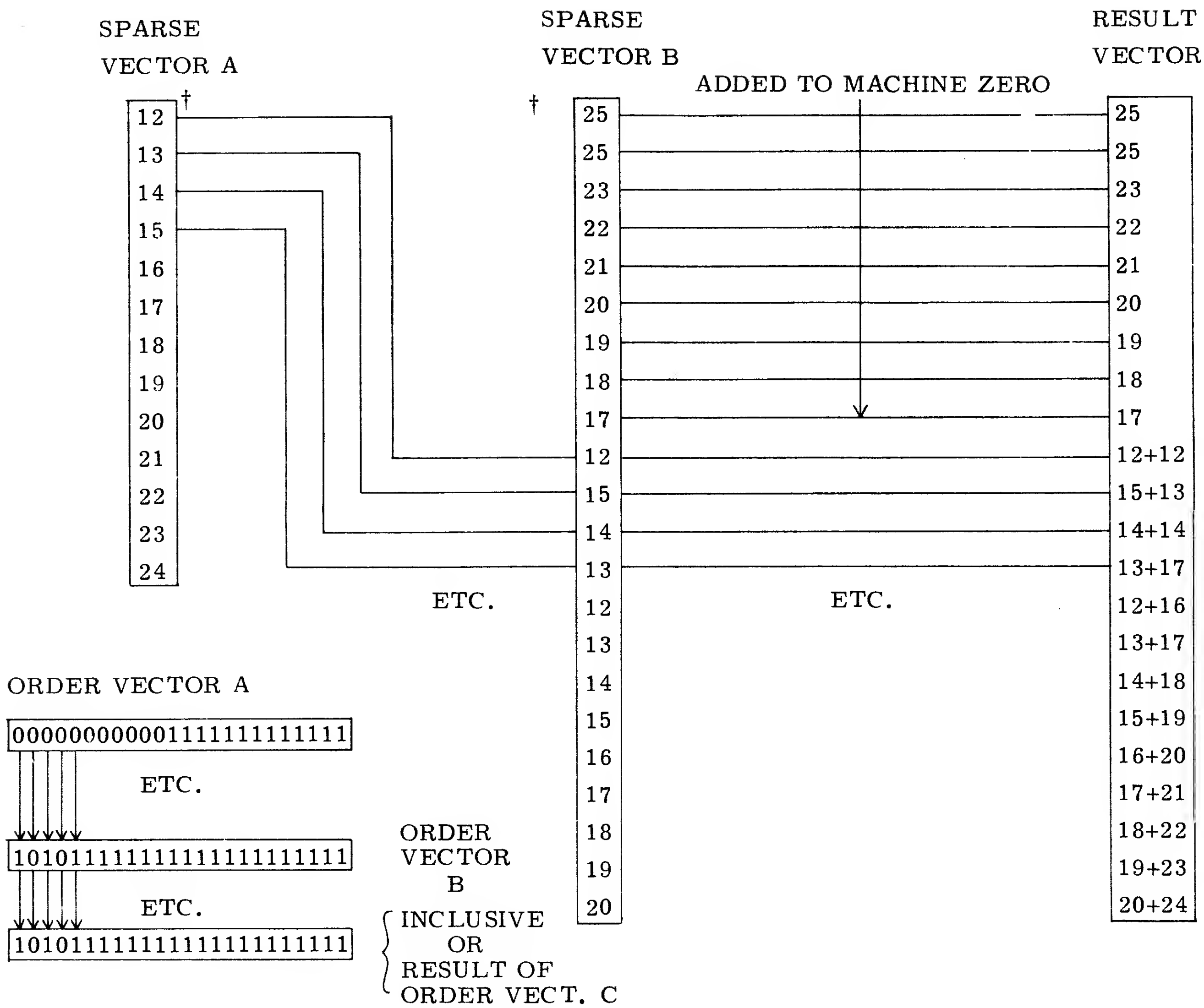
13 14 15 16 17 18 19 20
21 22 23 24 ⟶

Compressed Matrix B

25 25 23 22 21 20 19 18
17 12 15 14 13 12 13 14
15 16 17 18 19 20 ⟶

These matrices, in abbreviated form, are summed and the inclusive OR results of their order vectors are placed in a register. The inclusive OR forms the order vector for the resultant sparse vector. The following figure provides a functional view of the ADDNS instruction.

Result of ADDNS is:



†These values are normalized before the addition occurs and results are in normalized form.

```
                                                     .
                                                     TITLE "SPARSE VECTOR ADD"

                                                     INPUT 1,80,26
                                                     OUTPUT
                                                     IDENT
02 0000000000                                        PSEC 2
                                                     ENTRY START
                                                     ***REGISTER_DEFINITIONS
                         00 0000002840   REG_1       EQU #A1*64        * ORIGINAL MATRIX A DESCRIPTOR
                         00 0000002880   REG_2       EQU #A2*64        * ORIGINAL MATRIX B DESCRIPTOR
                         00 00000028C0   REG_3       EQU #A3*64        * BROADCASTMATRIX C DESCRIPTOR
                         00 0000002900   REG_4       EQU #A4*64        * ORDER VECTOR MATRIXA REG
                         00 0000002940   REG_5       EQU #A5*64        * ORDER VECTOR MATRIXB REG
                         00 0000002980   REG_6       EQU #A6*64        * COMPRESSED MATRIX A DESCRIPTOR
                         00 00000029C0   REG_7       EQU #A7*64        * COMPRESSED MATRIX B DESCRIPTOR
                         00 0000002A00   REG_8       EQU #A8*64        * MATRIXC DESCRIPTOR REG (RESULT)
                         00 0000002A40   REG_9       EQU #A9*64        * RESULT INCLUSIVE OR ORDER VECTOR
                         00 0000002800   DBR         EQU #A0*64        *DATA BASE REG
                                                     *
                         00 0000000540   VITAL       EQU #15*64
                         00 0000000680   RTN         EQU #1A*64
                         00 00000006C0   DSP         EQU #1B*64
                         00 0000000700   CSP         EQU #1C*64
                         00 0000000740   PSP         EQU #1D*64
                         00 0000000780   CLB         EQU #1E*64
                         00 00000007C0   UNIT        EQU #1F*64
02 0000000000 F                          START
02 0000000000 F 381C0015                             LTCL CSP,VITAL
02 0000000020 H 93000015 0000001C                    VTCV VITAL,CSP
02 0000000060 H 731C001D                             RTCR CSP,PSP
02 0000000080 F 731B001C                             RTCR DSP,CSP
02 00000000A0 H 3F1B0000                             IS DSP,0
                                                     *
02 00000000C0 F 781E00A0                             RTCR CLB,DBR
                                                     *
                                                     ***GENERATE DESCRIPTORS IN REGISTERS
02 00000000E0 H 2AA00009                             ELEN DBR,9          *ENTER LENGTH INTO DATA BASE REGISTER
02 0000000100 F BEA10000 00002840                    EX REG_1,REG_1          *SET POINTER FOR REG_1
02 0000000140 F 2AA10009                             ELEN REG_1,9          *SET POINTER LENGTH
02 0000000160 H 930000A0 000000A1                    VTCV DBR,REG_1          *VECT TO VECT TRAS MATRIX LOC TO REGS
02 00000001A0 H C60800A1 00A3A400   COMPA            CMPGE,E REG_1,REG_3,REG_4      *CREATE ORDER VECTOR
02 00000001E0 H C60800A2 00A3A500   COMPB            CMPGE,E REG_2,REG_3,REG_5      *CREATE ORDER VECT
                                                     *
                                                     **COMPRESS TO SPARSE VECTORS**
                                                     *
02 0000000220 H BC0000A1 0000A4A6   RESULT1          CPSV REG_1,REG_6,REG_4
02 0000000260 H BC0000A2 0000A5A7   RESULT2          CPSV REG_2,REG_7,REG_5
02 00000002A0 H A200A4A6 A5A7A9A8   ADDITION         ADDNS [REG_6,REG_4],[REG_7,REG_5],[REG_8,REG_9]
02 00000002E0 H 381D0015                             LTCL PSP,VITAL
02 0000000300 F 9800001D 00000015                    VTCV PSP,VITAL
02 0000000340 F 3340001A                             EXCF,BR ,RTN
03 0000000000                                        PSEC
                                                     ****DESCRIPTOR SETUP
                                    PRESET           FORM,64 16,48
03 0000000000 F 0018                MATRIXA          PRESET FLO_LT,START_A          *DESCRIPTOR OR MTRIXA
03 0000000010 C 000000000240(03)
03 0000000040 F 0018                MATRIXB          PRESET FLO_LT,START_B          *DESCRIPTOR FOR MATRIXB
03 0000000050 C 000000000880(03)
```

```
CCC STAR ASSEMBLER VER 2.2.2          SPARSE VECTOR ADD                                    DATE: 17SEP74    PAGE    3

03 0000000080 F 00000000 0000000C     MATRIXC  PRESET 0,START_C               *DESCRIPTOR FR MTRIXC              1/0059
03 00000000C0 F 0018                  O_VECT   PRESET FLD_LT,ORDER_VECT        *ORDER VECTOR FOR MATRIXA         1/0060
03 00000000D0 C 000000000E80(03)
03 0000000100 F 0018                  O2_VECT  PRESET FLD_LT,ORDER_VECT2       *ORDER VECTOR MATRIXB             1/0061
03 0000000110 C 0000000017E0(03)
03 0000000140 F 000C                  SPARSA   PRESET FLD_LT/2,RESULTA         *DESCRIPTOR FOR SPARSEA           1/0062
03 0000000150 C 000000002AC0(03)
03 0000000180 F 000C                  SPARSE   PRESET FLD_LT/2,RESULTB         *DESCRIPTOR FOR SPARSEB           1/0063
03 0000000190 C 000000003440(03)
03 00000001C0 F 000C                  SPARSC   PRESET FLD_LT/2,RESULTC         *DESCRIPTOR FOR SPARSEC           1/0064
03 00000001D0 C 0000000030C0(03)
03 0000000200 F 0018                  O3_VECT  PRESET FLD_LT,ORDER_VECT3          *ORDER ECT RESULT              1/0065
03 0000000210 C 000000002140(03)
                        00 0000000018 FLD_LT   EQU 24                *FIELD_LENGTH O ALL MATRICIES               1/0066
                                               **GENERATE MATRICIES***                                           1/0067
03 0000000240 F 00000000 00000001     START_A  GEN 1,2,3,4,4,5,6,7,8          *ROW1-MATRIXA                      1/0068
03 0000000280 F 00000000 00000002
03 00000002C0 F 00000000 00000003
03 0000000300 F 00000000 00000004
03 0000000340 F 00000000 00000004
03 0000000380 F 00000000 00000005
03 00000003C0 F 00000000 00000006
03 0000000400 F 00000000 00000007
03 0000000440 F 00000000 00000008
03 0000000480 F 00000000 00000009              GEN 9,10,11,12,13,14,15,16        *ROW2-MATRIXA                  1/0069
03 00000004C0 F 00000000 0000000A
03 0000000500 F 00000000 0000000B
03 0000000540 F 00000000 0000000C
03 0000000580 F 00000000 0000000D
03 00000005C0 F 00000000 0000000E
03 0000000600 F 00000000 0000000F
03 0000000640 F 00000000 00000010
03 0000000680 F 00000000 00000011               GEN 17,18,19,20,21,22,23,24        *ROW3=MATRIXA                 1/0070
03 00000006C0 F 00000000 00000012
03 0000000700 F 00000000 00000013
03 0000000740 F 00000000 00000014
03 0000000780 F 00000000 00000015
03 00000007C0 F 00000000 00000016
03 0000000800 F 00000000 00000017
03 0000000840 F 00000000 00000018
                                               *****************************                                     1/0071
03 0000000880 F 00000000 00000019     START_B  GEN 25,11,25,10,23,22,21,20        *ROW1-MATRIXE                  1/0072
03 00000008C0 F 00000000 0000000B
03 0000000900 F 00000000 00000019
03 0000000940 F 00000000 0000000A
03 0000000980 F 00000000 00000017
03 00000009C0 F 00000000 00000016
03 0000000A00 F 00000000 00000015
03 0000000A40 F 00000000 00000014
03 0000000A80 F 00000000 00000013              GEN 19,18,17,12,15,14,13,12        *ROW2-MATRIXE                  1/0073
03 0000000AC0 F 00000000 00000012
03 0000000B00 F 00000000 00000011
03 0000000B40 F 00000000 0000000C
03 0000000B80 F 00000000 0000000F
03 0000000BC0 F 00000000 0000000E
03 0000000C00 F 00000000 0000000D
03 0000000C40 F 00000000 0000000C
```

```
 CDC STAR ASSEMBLER VER 2.2.2          SPARSE VECTOR ADD                                                DATE: 17SEP74    PAGE     4

03 00000000C80 F 00000000 0000000D               GEN 13,14,15,16,17,18,19,20           *ROW3-MATRIX3                 1/0074
03 00000000CC0 F 00000000 0000000E
03 00000000D00 F 00000000 0000000F
03 00000000D40 F 00000000 00000010
03 00000000D80 F 00000000 00000011
03 00000000DC0 F 00000000 00000012
03 00000000E00 F 00000000 00000013
03 00000000E40 F 00000000 00000014
                      00 0000000000C    START_C  EQU #C                                                                   1/0075
03 00000000E80                          ORDER_VECT RES #64*24          *RESERVE FOR MATRIX_A O_VECTOR                    1/0076
03 000000017E0                          ORDER_VECT2 RES #64*24          *RESERVE FOR MTRIX_B O_VECTOR                    1/0077
03 00000002140                          ORDER_VECT3 RES #64,*24                                                           1/0078
03 00000002AC0                          RESULTA  RES,64 #64*24                                                            1/0079
03 00000003440                          RESULTB  RES,64 #64*24                                                            1/0080
03 00000003DC0                          RESULTC  RES,64 #64*24                                                            1/0081
                                                 END START                                                                1/0082


 CDC STAR ASSEMBLER VER 2.2.2          SPARSE VECTOR ADD                                                DATE: 17SEP74    PAGE     5

NUMBER OF WARNING MESSAGES =      0
NUMBER OF ERROR MESSAGES   =      0


CDC STAR ASSEMBLER VER 2.2.2                                                                          DATE: 17SEP74    PAGE     6

                                               FINIS                                                                   1/0083


           ASSEMBLY FINISHED

         10:15 A.M. TUESDAY 17TH. SEPTEMBER, 1974.

         NUMBER OF STATEMENTS PROCESSED     83

         NUMBER OF WARNING MESSAGES   NONE

         NUMBER OF ERROR MESSAGES   NONE
```

```
                                                 TITLE "SPARSE VECTOR ADD"
```



```
                                                 INPUT 1,80,20
                                                   OUTPUT
                                                 IDENT
02 000000000000                                  MSEC  2
                                                     ENTRY   START
                                                 ***REGISTER_DEFINITIONS
                    0C 000000002840     REG_1  EQU #A1*64     * ORIGINAL MATRIX A DESCRIPTOR
                    0C 000000002880     REG_2  EQU #A2*64     * ORIGINAL MATRIX B DESCRIPTOR
                    0C 0000000028C0     REG_3  EQU #A3*64     * BROADCASTMATRIX C DESCRIPTOR
                    0C 000000002900     REG_4  EQU #A4*64     * ORDER VECTOR MATRIXA  REG
                    0C 000000002940     REG_5  EQU #A5*64     * ORDER VECTOR MATRIXB  REG
                    0C 000000002980     REG_6  EQU #A6*64     * COMPRESSED MATRIX A DESCRIPTOR
                    0C 0000000029C0     REG_7  EQU #A7*64     * COMPRESSED MATRIX B DESCRIPTOR
                    0C 000000002A00     REG_8  EQU #A8*64     * MATRIXC DESCRIPTOR REG (RESULT)
                    0C 000000002A40     REG_9  EQU #A9*64     * RESULT INCLUSIVE OR ORDER VECTOR
                    0C 000000002B00     DBR      EQU    #AC*64       *DATA BASE REG
                                        *
                    0C 000000000540     VITAL  EQU #15*64
                    0C 000000000680     RTN    EQU #1A*64
                    0C 0000000006C0     OSP   EQU #1B*64

                    0C 000000000700     CSP   EQU #1C*64
                    0C 000000000740     PSP    EQU #1D*64
                    0C 000000000780     COB  EQU  #1E*64
                    0C 0000000007C0     UNIT   EQU #1F*64
02 000000000000 F                       START
02 000000000000 F   391C0015                  LTOL   CSP,VITAL
02 000000000020 H   98000015 0000001C            VTOV    VITAL,CSP
02 000000000060 H   791C001D                     RTOR   CSP,PSP
02 000000000080 F   791B001C                    RTOR   OSP,CSP
02 0000000000A0 H   3F1B0000                     IS OSP,0
                                        *
02 0000000000C0 F   781E00AC                      RTOR COB,DBR
                                        *
                                                 ***GENERATE DESCRIPTORS IN REGISTERS
02 0000000000E0 H   2AAC0009                       ELEN   DBR,9     *ENTER LENGTH INTO DATA BASE REGISTER
02 000000000100 F   8EA10000 00002840              EX      REG_1,REG_1     *SET POINTER FOR REG_1
02 000000000140 F   2AA10009                       ELEN    REG_1,9     *SET POINTER  LNGTH
02 000000000160 H   9900000AC 000000A1             VTOV   DBR,REG_1    *VECT TO VECT TRAS MATRIX LOC TO REGS
02 0000000001A0 H   C60800A1 00A3A400   COMPA    CMPGE,B   REG_1,REG_3,REG_4   *CREATE ORDER VECTOR
02 0000000001E0 H   C60800A2 00A3A500   COMPB    CMPGE,B    REG_2,REG_3,REG_5   *CREATE ORDER VECT
                                                   *
                                                       **COMPRESS TO SPARSE VECTORS**
                                                   *
02 000000000220 H   9C0L00A1 0C00A4A6   RESULT1 CPSV REG_1,REG_6,REG_4
02 000000000260 H   9C0C00A2 0000A5A7   RESULT2 CPSV REG_2,REG_7,REG_5
02 0000000002A0 H   A20CA4A6 A5A7A9A8   ADDITION ADDNS (REG_6,REG_4),(REG_7,REG_5),(REG_8,REG_9)
02 0000000002E0 H   39100015                       LTOL PSP,VITAL
02 000000000300 F   99000010 00000015              VTOV PSP,VITAL
02 000000000340 F   334C001A                       BAOF,BR  ,RTN

03 000000000000                                    MSEC
                                                 ****DESCRIPTOR SETUP
                                        PRESET  FORM,64 16,48
03 000000000000 F   0018                MATRIXA PPESET FLO_LT,START_A          *DESCRIPTOR FOR MATRIXA
03 000000000010 C   000C00000240(03)
03 000000000040 F   0018                MATRIXB PRESET FLO_LT,START_B          *DESCRIPTOR FOR MATRIXB
03 000000000050 C   000C00000880(03)
03 000000000080 F   000L0000 0000000LC  MATRIXC PPESET 0,START_C               *DESCRIPTOP FOR MATRIXC
```

```
CDC STAR ASSEMBLER  VER 1.7          SPARSE VECTOR ADD                                        DATE: 18APR73   PAGE   3

03 0000000000C0 F    0018                  O_VECT PRESET FLD_LT,ORDER_VECT   *ORDER VECTOR FOR MATRIXA           1/0060
03 0000000000D0 C    000C00000E80(03)
03 000000000100 F    0018                  O2_VECT  PRESET  FLD_LT,ORDER_VECT2         *ORDER VECTOR MATRIXB     1/0061
03 000000000110 C    000C000017E0(03)
03 000000000140 F    000C                  SPARSA  PRESET FLD_LT/2,RESULTA       *DESCRIPTOR FOR SPARSEA         1/0062
03 000000000150 C    000C00002AC0(03)
03 000000000180 F    000C                  SPARSB  PRESET FLD_LT/2,RESULTB       *DESCRIPTOR FOR SPARSEB         1/0063
03 000000000190 C    000C00002F80(03)
03 0000000001C0 F    000C0000 05000000     SPARSC  PRESET FLD_LT/2,RESULTC       *DESCRIPTOR FOR SPARSEC         1/0064
03 000000000200 F    0018                  O3_VECT     PRESET     FLD_LT,ORDER_VECT3     *ORDER VECT RESULT      1/0065
03 000000000210 C    000C00002140(03)
                       0C 000000000018     FLD_LT  EQU 24                        *FIELD_LENGTH FOR ALL MATRICIES 1/0066
                                                   **GENERATE MATRICIES***                                       1/0067
03 000000000240 F    00000000 00000001     START_A GEN 1,2,3,4,4,5,6,7,8         *ROW1-MATRIXA                   1/0068
03 000000000280 F    00000000 00000002
03 0000000002C0 F    00000000 00000003
03 000000000300 F    00000000 00000004
03 000000000340 F    00000000 00000004
03 000000000380 F    00000000 00000005
03 0000000003C0 F    00000000 00000006
03 000000000400 F    00000000 00000007
03 000000000440 F    00000000 00000008
03 000000000480 F    00000000 00000009             GEN 9,10,11,12,13,14,15,16    *ROW2-MATRIXA                   1/0069
03 0000000004C0 F    00000000 0000000A
03 000000000500 F    00000000 0000000B
03 000000000540 F    00000000 0000000C
03 000000000580 F    00000000 0000000D
03 0000000005C0 F    00000000 0000000E
03 000000000600 F    00000000 0000000F
03 000000000640 F    00000000 00000010
03 000000000680 F    00000000 00000011             GEN 17,18,19,20,21,22,23,24   *ROW3-MATRIXA                   1/0070
03 0000000006C0 F    00000000 00000012
03 000000000700 F    00000000 00000013
03 000000000740 F    00000000 00000014
03 000000000780 F    00000000 00000015
03 0000000007C0 F    00000000 00000016
03 000000000800 F    00000000 00000017
03 000000000840 F    00000000 00000018
                                                   ****************************                                  1/0071
03 000000000880 F    00000000 00000019     START_B GEN 25,11,25,10,23,22,21,20   *ROW1-MATRIXB                   1/0072
03 0000000008C0 F    00000000 0000000B
03 000000000900 F    00000000 00000019
03 000000000940 F    00000000 0000000A
03 000000000980 F    00000000 00000017
03 0000000009C0 F    00000000 00000016
03 000000000A00 F    00000000 00000015
03 000000000A40 F    00000000 00000014
03 000000000A80 F    00000000 00000013             GEN 19,18,17,12,15,14,13,12   *ROW2-MATRIXB                   1/0073
03 000000000AC0 F    00000000 00000012
03 000000000B00 F    00000000 00000011
03 000000000B40 F    00000000 0000000C
03 000000000B80 F    00000000 0000000F
03 000000000BC0 F    00000000 0000000E
03 000000000C00 F    00000000 0000000D
03 000000000C40 F    00000000 0000000C
03 000000000C80 F    00000000 0000000D             GEN 13,14,15,16,17,18,19,20   *ROW3-MATRIXB                   1/0074
03 000000000CC0 F    00000000 0000000E


CDC STAR ASSEMBLER  VER 1.7          SPARSE VECTOR ADD                                        DATE: 18APR73   PAGE   4

03 000000000D00 F    00000000 0000000F
03 000000000D40 F    00000000 00000010
03 000000000D80 F    00000000 00000011
03 000000000DC0 F    00000000 00000012
03 000000000E00 F    00000000 00000013
03 000000000E40 F    00000000 00000014
                       0C 00000000000C     START_C   EQU   #C                                                    1/0075
03 000000000E80                            ORDER_VECT  RES #64*24        *RESERVE FOR MATRIX_A O_VECTOR          1/0076
03 0000000017E0                            ORDER_VECT2 RES #64*24        *RESERVE FOR MATRIX_B O_VECTOR          1/0077
03 000000002140                            ORDER_VECT3 RES #64*24                                                1/0078
03 000000002AC0                            RESULTA RES,64  #64*12                                                1/0079
           2F80                            RESULTB RES,64  #64*12                                                1/0080
                                                                                                                 1/0081
                                                   END  START                                                    1/0082
```

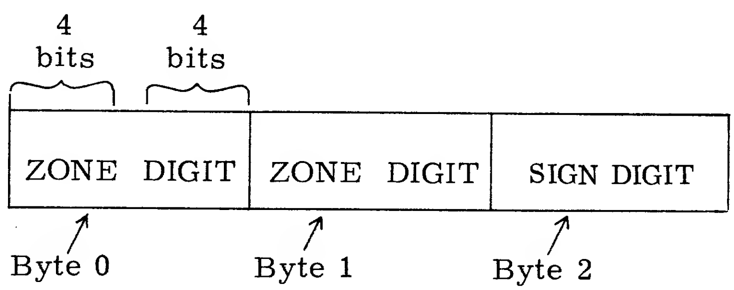
```
COC STAR ASSEMBLER  VER 1.7          SPARSE VECTOR ADD                                         DATE: 18APR73   PAGE    5

NUMBER OF WARNING MESSAGES =     0
NUMBER OF ERROR MESSAGES   =     0


COC STAR ASSEMBLER  VER 1.7                                                                    DATE: 18APR73   PAGE    6
                                              FINIS                                                                  1/C083


        ASSEMBLY FINISHED

      3:40 A.M. WEDNESDAY 18TH. APRIL, 1973.

      NUMBER OF STATEMENTS PROCESSED   158

      NUMBER OF WARNING MESSAGES  NONE

      NUMBER OF ERROR MESSAGES  NONE


   STAR LOADER  V1.1           3:42 A.M. WEDNESDAY 18TH. APRIL, 1973.

                                                          PLA 1             03:40:04.295  18/04/73
                                          CODE          000001000000        000380
                                          OATA          000001000380        003440
      START                               ENTRY         000001000060

 TOTAL ELAPSED TIME FOR THIS LOAO WAS 4 SECONDS.

 1 PAGE(S) OF OATA WERE ALLOCATED.

 1 MODULES OEFINING 1 SYMBOLS WERE LOAOED.


*** OUMP OF VIRTUAL MEMORY FROM ADDRESS 000005000000 TO 000005020000

                                                 OUTPUT

000005000000    FFD66400 00000000  FFD66400 00000000  FFD65C00 00000000  FFD65800 00000000    .................\........X.....
000005000100    FFD65400 00000000  FFD65000 00000000  FFD64C00 00000000  FFD64800 00000000    ..T........P........L........H.....
000005000200    FFD64400 00000000  FFD56000 00000000  FFD66C00 00000000  FFD66C00 00000000    ..D............................
000005000300    FFD66C00 00000000  FFD66000 00000000  FFD67400 00000000  FFD67C00 00000000    ...............................
000005000400    FFD74200 00000000  FFD74600 00000000  FFD74A00 00000000  FFD74E00 00000000    ..B........F........J........N.....
000005000500    FFD75200 00000000  FFD75600 00000000  00000000 00000000  00000000 00000000    ..R........V.......................
000005000500    00000000 00000000  00000000 00000000  00000000 00000000  00000000 00000000    ...............................
*** RPT. ***
000005008000    00000000 00001F1C  00000000 00000000  00000000 00000000  00000000 00000000    ...............................
000005008100    00000000 00000000  00000000 00000000  00000000 00000000  00000000 00000000    ...............................
*** RPT. ***

000005010000    00000000 00001F1C  00000000 00000000  00000000 00000000  00000000 00000000    ...............................
000005010100    00000000 00000000  00000000 00000000  00000000 00000000  00000000 00000000    ...............................
*** RPT. ***
000005018000    00000000 00001F1C  00000000 00000000  00000000 00000000  00000000 00000000    ...............................
000005018100    00000000 00000000  00000000 00000000  00000000 00000000  00000000 00000000    ...............................
*** RPT. ***
000005020000    00000000 00001F1C  00000000 00000000  00000000 00000000  00000000 00000000    ...............................

 *** ENO OF VIRTUAL MEMORY DUMP
```

STRING INSTRUCTIONS

String instructions perform arithmetic and logical operations on strings of data in the form of 8-bit bytes. The byte size allows for handling large alphabets (256 characters) and is compatible with ASCII extended binary code. The field length of a data string can be extended beyond one 64-bit word or can be less than one data word. Bytes in the field of a data string are in opposite order of the byte address; the most significant byte is the leftmost byte, but, the address of the leftmost byte is 0.

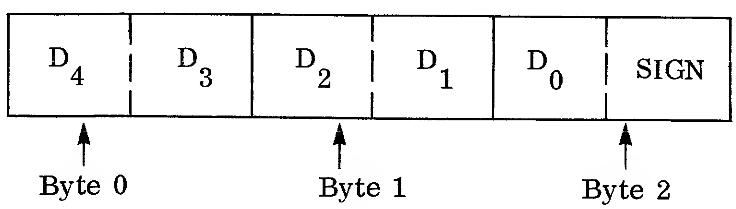
Unless specified by the instruction, strings are processed from right to left until the last byte in the field is processed. Normally, string instructions terminate when the result field is filled.

String instructions perform operations on data strings in packed binary-coded decimal (BCD) form, zoned BCD, and binary formats. The zoned-decimal format is used for I/O operations. Each byte, with exception of the rightmost byte, contains a BCD digit with a zone designator (3) located in the leftmost 4 bits of each byte. The rightmost byte contains the sign in the leftmost 4 bits. (A for +, B for -.)



ZONED BCD

The packed decimal form normally is used for arithmetic operations. The rightmost 4 bits of the rightmost byte contain the sign, the remaining bytes consist of two 4-bit digits.



PACKED BCD

Binary numbers are represented in strings of 8-bit bytes. The leftmost bit of the leftmost byte contains a sign (0 for +) (1 for -). All binary numbers are sign extended through the sign bit. All negative numbers are two's complement.

String instructions make use of string indexes, which are item counts in bytes, for all instructions with the exception of D6 and FF. A string index can have a value of up to 2^{45-1}. The leftmost 3 bits of a string index are not used, the sign of a negative index is extended through bit 16, and overflow is not detected when an index is added to a base address.

DELIMITERS

There are six string instructions which permit delimiter termination: these are F8, F9, FD, EE, EF, and D7. All other string instructions have length limited fields. Delimiters are contained in bits 0 through 15 of a designated register. When a character in the data field location matches the delimiter value, the instruction terminates. Field length or delimiter character is selected by G designator bits.

Bits	
d (8 and 9)	Designator for A and B
e (10 and 11)	Designator for C
(12 and 14)	Undefined 0's
(13 and 15)	Increment A field and C field index respectively

Table C-5. String Instruction G Designators

Designator	d/e Bit Value	Function
d and/or e	00	The 16-bit length specification in A, B, and/or C represents an item count of the number of bytes or bits in the field (field length).
d and/or e	10	The rightmost 8 bits of the length specification in A, B, and/or C are used as a delimiter character.
d and/or e	11	The entire 16 bits of the length specification in A, B, and/or C are used as a delimiter character.
d	01	The rightmost 8 bits of the length specification function as a delimiter character. The leftmost 8 bits serve as a mask on the comparison. Bits in the delimiter character and the operand byte are compared only where 1's exist in the mask. This specification applies only to source fields. Any instruction becomes undefined if this specification is used for a result field.

INCREMENTS

Nine instructions use index incrementing: F8, F9, FD, FE, D6, D7, EE, EF, and FF. At the termination of these instructions, the index register fields are left in one of the following states:

No Increment – The index register remains at its original value. An example is the index register associated with a translate table. Characters to be translated are added to the indexed address of the table to obtain the translated character. The index associated with the table does not change during the instruction execution.

Partial Increment – The index register is incremented to specify a particular character or word in its associated field. An example is the FD instruction which searches two byte strings for inequality. When an inequality is found, the search terminates and a count equal to the number of no-hit comparison is added to each index. The end may not have reached field lengths, but the location of the unequal characters can be formed by manipulating the incremented index and the base address.

Full Increment – The index register is incremented by the full length of its associated field. For example, when the translate instruction is terminated, the index associated with source field A is incremented by the length of field A to specify the starting bit of the next contiguous field. If field length is specified by a delimiter character, the field is searched for that character. The index of the associated field is incremented then so the starting point is one character beyond the delimiter characters.

LOGICAL STRING INSTRUCTIONS

These instructions function in the same general manner as corresponding string instructions. They operate with index and data fields the same as for string instructions except item counts are expressed in bits instead of bytes; therefore, these instructions perform bit operations on bit boundaries.

MONITOR INSTRUCTIONS

The monitor instructions function only during monitor mode. When a machine is in job mode, any attempt to execute a monitor instruction is detected by the hardware as an attempt to perform an undefined function code.

NON-TYPICAL INSTRUCTIONS

These instructions perform operations such as register to storage transfers; formation of repeated mask lists; and maximum/minimum determinations that do not belong in any of the preceding instruction types discussed.

SIGN CONTROL

Certain vector, sparse vector, and non-typical instructions provide an operation called sign control on the input operands. (Table C-6.) For these instructions, bits 5, 6, and 7 of the G field have the following significance.

Bit 5	Bit 6	
0	0	Use the operands from the A stream in the normal manner.
0	1	Complement the coefficients of the operands from the A stream before using them.
1	0	Use the magnitude of the coefficients of the operands from the A stream.
1	1	Make all positive coefficients of the operands from the A stream negative before using them. Negative operands will not be altered.

Bit 7	
0	Use the operands from the B stream in the normal manner.
1	Use the magnitude of the coefficients of the operands from the B stream.

Any complementing necessary to achieve the required operand state is a 48-bit two's complement operation performed before operands are used in the specified arithmetic operation. If the complement of the coefficient 2000 0000 0000 is required, the operand will be used as 7000 0000 0000 with one added to its exponent, which could cause exponent overflow.

Any significance calculation necessary in performing an instruction is made before complementing occurs.

Table C-6. Instructions with Sign Control

Instruction		A Operands Bit 5 and Bit 6		B Operands (Bit 7)
80, 81, 82	Vector Add	X	X	X
84, 85, 86	Vector Subtract	X	X	X
88, 89, 8B	Vector Multiply	X	X	X
8C, 8F	Vector Divide	X	X	X
93	Vector Square Root	X	X	0
A0, A1, A2	Sparse Vector Add	X	X	X
A4, A5, A6	Sparse Vector Subtract	X	X	X
A8, A9, AB	Sparse Vector Multiply	X	X	X
AC, AF	Sparse Vector Divide	X	X	X
CF	Arithmetic Compress	X	X	X
D8	Maximum of A to C	X	0	0
D9	Minimum of A to C	X	0	0

X 0 or 1 bit is legal
0 This bit must always be set to zero

MACHINE INSTRUCTIONS

Tables C-7 through C-17 list all of the machine instructions available with the Control Data STAR computer system. They include:

- Instruction OP Code
- Format (F)
- Instruction Mnemonic
- Applicable Operands
- Applicable Qualifiers

Register designators contained in the operand portion of the table are defined in table C-17.

Table C-7. Index Instructions

Op	F	Mnemonic	Qualifiers	Operands	Description
3E	6	ES	none	R_f,I16	Enter short, full word: I16 → $R_{16\text{-}63}$, R.J., S E; 0 → $R_{0\text{-}15}$
4D	6	ESH		R_h,I16	Enter short, half-word: I16 → $R_{8\text{-}31}$, R.J., S E; 0 → $R_{0\text{-}7}$
BE	5	EX		R_f,I48	Enter index, full word: I48 → $R_{16\text{-}63}$, 0 → $R_{0\text{-}15}$
CD	5	EXH		R_h,I24	Enter index, half-word: I24 → $R_{8\text{-}31}$, 0 → $R_{0\text{-}7}$
3F	6	IS		R_f,I16	Increase short, full word: $R_{16\text{-}63}$ + I16 → $R_{16\text{-}63}$, $R_{0\text{-}15}$ unchanged
4E	6	ISH		R_h,I16	Increase short, half-word: I16 + $R_{8\text{-}31}$ → $R_{8\text{-}31}$, $R_{0\text{-}7}$ unchanged
BF	5	IX		R_f,I48	Increase index, full word: I48 + R → R
CE	5	IXH		R_h,I24	Increase index, half-word: I24 + R → R
38	A	LTOL	none	R_L,T_L	Transmit length $R_{0\text{-}15}$ to length $T_{0\text{-}15}$, $T_{16\text{-}63}$ unchanged

Table C-8. Register Instructions

Op	F	Mnemonic	Qualifiers	Operands	Description
79	A	ABS	none	R_f,T_f	Absolute, full word F P: $ABS(R_f) \rightarrow T_f$
59	A	ABSH		R_h,T_h	Absolute, half-word F P: $ABS(R_h) \rightarrow T_h$
61	4	ADDL		R_f,S_f,T_f	Add lower, full word F P: $(R_f) + (S_f))_L \rightarrow T_f$
2B	4	ADDLEN		R_L,S_f,T_L	Add to length: $R_{0\text{-}15} + S_{40\text{-}63} \rightarrow T_{0\text{-}15}, R_{16\text{-}63} \rightarrow T_{16\text{-}63}$
41	4	ADDLH		R_h,S_h,T_h	Add lower, half-word F P: $((R_h) + (S_h))_L \rightarrow T_h$
62	4	ADDN		R_f,S_f,T_f	Add normalized, full word F P: $((R_f) + (S_f))_n \rightarrow T_f$
42	4	ADDNH		R_h,S_h,T_h	Add normalized, half-word F P: $((R_h) + (S_h))_n \rightarrow T_h$
60	4	ADDU		R_f,S_f,T_f	Add upper, full word F P: $((R_f) + (S_f))_u \rightarrow T_f$
40	4	ADDUH		R_h,S_h,T_h	Add upper, half-word F P: $((R_h) + (S_h))_u \rightarrow T_h$
63	4	ADDX		R_f,S_f,T_f	Add index (address), full word: $R_{16\text{-}63} + S_{16\text{-}63} \rightarrow T_{16\text{-}63}, R_{0\text{-}15} \rightarrow T_{0\text{-}15}$
75	4	ADJE		R_f,S_f,T_f	Adjust exponent, full word F P: (R_f) per $S \rightarrow T_f$
55	4	ADJEH		R_h,S_h,T_h	Adjust exponent, half-word F P: (R_h) per $S \rightarrow T_h$
74	4	ADJS		R_f,S_f,T_f	Adjust significance (shift), full word F P: (R_f) per $S \rightarrow T_f$
54	4	ADJSH		R_h,S_h,T_h	Adjust significance (shift), half-word F P: (R_h) per $S \rightarrow T_h$
11	A	BTOD		R_f,T_f	Convert binary R to packed BCD T, fixed length
72	A	CLG		R_f,T_f	Ceiling, full word F P: nearest integer .GE. $(R_f) \rightarrow T_f$
52	A	CLGH		R_h,T_h	Ceiling, half-word F P: nearest integer .GE. $(R_h) \rightarrow T_h$
76	A	CON	none	R_f,T_h	Contract, full word F P: $R_{64} \rightarrow T_{32}$

Table C-8. Register Instructions (Cont'd)

Op	F	Mnemonic	Qualifiers	Operands	Description
6F	4	DIVS	none	R_f,S_f,T_f	Divide significant, full word F P: $((R_f)/(S_f))_s \rightarrow T_f$
4F	4	DIVSH		R_h,S_h,T_h	Divide significant, half-word F P: $((R_h)/(S_h))_s \rightarrow T_h$
6C	4	DIVU		R_f,S_f,T_f	Divide upper, full word F P: $((R_f)/(S_f))_u \rightarrow T_b$
4C	4	DIVUH		R_h,S_h,T_h	Divide upper, half-word F P: $((R_h)/(S_h))_u \rightarrow T_h$
10	A	DTOB		R_f,T_f	Convert packed BCD to binary T fixed length
2A	6	ELEN		R_L,I16	Enter length: I16 $\rightarrow R_{0\text{-}15}$, $R_{16\text{-}63}$ unchanged
7A	A	EXP		R_L,T_f	Exponent, full word: $R_{0\text{-}15} \rightarrow T_{16\text{-}63}$, S E, $0 \rightarrow T_{0\text{-}15}$
5A	A	EXPH		R_{Lh},T_h	Exponent, half-word: $R_{0\text{-}7} \rightarrow T_{8\text{-}31}$, S E, $0 \rightarrow T_{0\text{-}7}$
6E	4	EXTB		R_f,S_d,T_f	Extract bits from R_f to T_f per S_d
5C	A	EXTH		R_h,T_f	Extend half-word F P: $R_{32} \rightarrow T_{64}$
5D	A	EXTXH		R_h,T_f	Extend index, half-word F P: $R_{8\text{-}31} \rightarrow T_{16\text{-}63}$,S E, $R_{0\text{-}7} \rightarrow T_{0\text{-}15}$,S E
71	A	FLR		R_f,T_f	Floor, full word F P: nearest integer .LE. $(R_f) \rightarrow T_f$
51	A	FLRH		R_h,T_h	Floor, half-word F P: nearest integer .LE. $(R_h) \rightarrow T_h$
6D	4	INSB		R_f,S_d,T_f	Insert bits from R_f to T_f per S_d
7C	A	LTOR		R_L,T_f	Length to register, full word F P: $R_{0\text{-}15} \rightarrow T_{48\text{-}63}$, $0 \rightarrow T_{0\text{-}47}$
69	4	MPYL		R_f,S_f,T_f	Multiply lower, full word F P: $((R_f)*(S_f))_L \rightarrow T_f$
49	4	MPYLH		R_h,S_h,T_h	Multiply lower, half-word F P: $((R_h)*(S_h))_L \rightarrow T_h$
6B	4	MPYS		R_f,S_f,T_f	Multiply significant, full word F P: $((R_f)*(S_f))_s \rightarrow T_f$
4B	4	MPYSH		R_h,S_h,T_h	Multiply significant, half-word F P: $((R_h)*(S_h))_s \rightarrow T_h$
68	4	MPYU	none	R_f,S_f,T_f	Multiply upper, full word F P: $((R_f)*(S_f))_u \rightarrow T_f$

Table C-8. Register Instructions (Cont'd)

Op	F	Mnemonic	Qualifiers	Operands	Description
48	4	MPYUH	none	R_h,S_h,T_h	Multiply upper, half-word F P: $((R_h)*(S_h))_u \rightarrow T_h$
7B	4	PACK		R_f,S_f,T_f	Pack, full word F P: $R_{48\text{-}63}$ & $S_{16\text{-}63} \rightarrow T_f$ R: exponent
5B	4	PACKH		R_h,S_h,T_h	Pack, half-word F P: $R_{24\text{-}31}$ & $S_{8\text{-}31} \rightarrow T_h$ S: coefficient
2D	4	RAND		R_f,S_f,T_f	Logical AND R,S, to T
77	A	RCON		R_f,T_h	Rounded contract, full word F P: $R_{64} \rightarrow T_{32}$
2E		RIOR		R_f,S_f,T_f	Logical inclusive OR R, S, to T
78	A	RTOR		R_f,T_f	Register to register full word transmit: $(R_f) \rightarrow T_f$
58	A	RTORH		R_h,T_h	Register to register half-word transmit: $(R_h) \rightarrow T_h$
2C	4	RXOR		R_f,S_f,T_f	Logical exclusive OR R, S, to T
34	4	SHIFT			Shift R_f by (S_f) to T_f
30	7	SHIFTI		R_f,I_8,T_f	Shift R_f by I_8 to T_f
73	A	SQRT		R_f,T_f	Significant square root, full word F P: $SQRT(R_f)_s \rightarrow T_f$
53	A	SQRTH		R_h,T_h	Significant square root, half-word, F P: $SQRT(R_h)_s \rightarrow T_h$
65	4	SUBL		R_f,S_f,T_f	Subtract lower, full word F P: $((R_f) - (S_f))_L \rightarrow T_f$
45	4	SUBLH		R_h,S_h,T_h	Subtract lower, half-word F P: $((R_h) - (S_h))_L \rightarrow T_f$
66	4	SUBN		R_f,S_f,T_f	Subtract normalized, full word F P: $((R_f) - (S_f))_n \rightarrow T_f$
46	4	SUBNH		R_h,S_h,T_h	Subtract normalized, half-word F P: $((R_h) - (S_h))_n \rightarrow T_f$
64	4	SUBU		R_f,S_f,T_f	Subtract upper, full word F P: $((R_f) - (S_f))_u \rightarrow T_f$
44	4	SUBUH		R_h,S_h,T_h	Subtract upper, half-word F P: $((R_h) - (S_h))_u \rightarrow T_h$
67	4	SUBX		R_f,S_f,T_f	Subtract index (address): $R_{16\text{-}63} - S_{16\text{-}63} \rightarrow T_{16\text{-}63}, R_{0\text{-}15} \rightarrow T_{0\text{-}15}$
7D		SWAP		R_d,S_f,T_d	Swap registers start with S_f; storing at T_d and loading from R_d
70	A	TRU		R_f,T_f	Truncate, full word F P: nearest integer .LE. $(R_f) \rightarrow T_f$
50	A	TRUH	none	R_h,T_h	Truncate, half-word F P: nearest integer .LE. $(R_h) \rightarrow T_h$

Table C-9. Branch Instructions

Op	F	Mnemonic	Qualifiers	Operands	Description
32	9	BAB	BR,BRO,BRZ, T,SO,SZ, BRB,BRF	S_a,T_a	Branch and alter bit: (S_a) is bit to be altered, (T_a) is branch address with qualifiers BRB & BRF branch address is relative ± I half-words
33	B	BADF	BR,BRO,BRZ, SO,SZ,T, BRB, BRF	$I6,T_a$	Data flag register bit branch and alter: I6 is bit altered (T_a) is branch address
2F	9	BARB	BR,BRO,BRZ T,SO,SZ	T,S	Branch to [S] on condition of bit 63 of register T
24	8	BEQ	none	R_f,S_f,T_a	Branch to (T_a) if (R_f) .EQ. (S_f), full word F P compare
26	8	BGE		R_f,S_f,T_a	Branch to (T_a) if (R_f) .GE. (S_f), full word F P compare
20	8	BHEQ			Branch to (T_a) if (R_h) .EQ. (S_h), half-word F P
22	8	BHGE		R_h,S_h,T_a	Branch to (T_a) if (R_h) .GE. (S_h), half-word F P compare
23	8	BHLT			Branch to (T_a) if (R_h) .LT. (S_h), half-word F P compare
21	8	BHNE			Branch to (T_a) if (R_h) .NE. (S_h), half-word F P compare
B6	5	BIM		R_i,I48	Branch immediate to (R_i) + I48
27	8	BLT		R_f,S_f,T_a	Branch to (T_a) if (R_f) .LT. (S_f), full word F P compare
25	8	BNE	none	R_f,S_f,T_a	Branch to (T_a) if (R_f) .NE. (S_f), full word F P compare

Table C-9. Branch Instructions (Cont'd)

Op	F	Mnemonic	Qualifiers	Operands	Description
36	7	BSAVE	none	R_f,[T_a,S_i]	Branch & save: set (R_f) to next instruction address, branch to [T_a + S_i]
35	7	DBNZ		R_f,[T_a,S_i]	Decrement & branch non-zero: (R_f)-1 → (R_f) if (R_f)≠0 branch to [T_a + S_i]
09	4	EXIT		none	Exit force, job to monitor
				S_a,T_a	Exit force, monitor to job, (S_a) register file, (T_a) invisible pkg
31	7	IBNZ		R_f,[T_a,S_i]	Increment & branch non-zero: (R_f) +1 → (R_f), if (R_f)≠ 0 branch to [T_a,S_i]
B0	C	IBXEQ	BAB,BRF		
B2	C	IBXGE		X_f,A_f,[B_a,Y_i], Z_f,C_f	Increment & branch index: $A_{16\text{-}63}$ + $X_{16\text{-}63}$ → $C_{16\text{-}63}$, $A_{0\text{-}15}$ → $C_{0\text{-}15}$
B5	C	IBXGT			if $A_{16\text{-}63}$ + $X_{16\text{-}63}$ .OP. $Z_{16\text{-}63}$ branch to (B_a) + (Y_i),
B4	C	IBXLE			or relative from the current location ± I16
B3	C	IBXLT		X_f,A_f,I16,Z_f,C_f	
B1	C	IBXNE			
3B	A	LSDFR	none	R_f,T_f	Load & store data flag register: (DFR) → T_f, (R_f) → DFR

Table C-10. Vector Instructions

Op	F	Mnemonic	Qualifiers	Operands	Description
99	1	ABSV	A,H,O,Z	[A,X] ,C,Z	Absolute vector: ABS(A) → C
81	1	ADDLV	A,B,C,H,MA, MB,N,O,Z	[A,X] ,[B,Y] ,C,Z	Add lower vector: $(A + B)_L \rightarrow C$
82	1	ADDNV	A,B,C,H,MA, MB,N,O,Z	[A,X] ,[B,Y] ,C,Z	Add normalized vector: $(A + B)_n \rightarrow C$
80	1	ADDUV	A,B,C,H,MA, MB,N,O,Z	[A,X] ,[B,Y] ,C,Z	Add upper vector: $(A + B)_u \rightarrow C$
83	1	ADDXV	A,B,O,Z	[A,X] ,[B,Y] ,C,Z	Add index vector: $A_{16\text{-}63} + B_{16\text{-}63} \rightarrow C_{16\text{-}63}$, $A_{0\text{-}15} \rightarrow C_{0\text{-}15}$
94	1	ADJSV	A,B,H,O,Z	[A,X] ,[B,Y] ,C,Z	Adjust significance vector: A per B → C
95	1	ADJEV	A,B,H,O,Z	[A,X] ,[B,Y] ,C,Z	Adjust exponent vector: A per B → C
92	1	CLGV	A,H,O,Z	[A,X] ,C,Z	Ceiling vector: nearest integer .GE. A → C
96	1	CONV	A,O,Z	[A,X] ,C,Z	Contract vector: $A_{64} \rightarrow C_{32}$
8C	1	DIVUV	A,B,C,H,MA, MB,N,O,Z	[A,X] ,[B,Y] ,C,Z	Divide upper vector: $(A/B)_u \rightarrow C$
8F	1	DIVSV	A,B,C,H,MA, MB,N,O,Z	[A,X] ,[B,Y] ,C,Z	Divide significant vector: $(A/B)_s \rightarrow C$
9A	1	EXPV	A,H,O,Z	[A,X] ,C,Z	Exponent vector: $A_{0\text{-}15} \rightarrow C_{48\text{-}63}$, S E, 0 → $C_{0\text{-}15}$
9C	1	EXTV	A,O,Z	[A,X] ,C,Z	Extend vector: $A_{32} \rightarrow C_{64}$

Table C-10. Vector Instructions (Cont'd)

Op	F	Mnemonic	Qualifiers	Operands	Description
91	1	FLRV	A,H,O,Z	[A,X] ,C,Z	Floor vector: nearest integer .LE. A $\rightarrow$ C
89	1	MPYLV	A,B,MA,MB, N,O,Z	[A,X] ,[B,Y] ,C,Z	Multiply lower vector: $(A*B)_L \rightarrow C$
8B	1	MPYSV	A,B,MA,MB, N,O,Z	[A,X] ,[B,Y] ,C,Z	Multiply significant vector: $(A*B)_s \rightarrow C$
88	1	MPYUV	A,B,MA,MB, N,O,Z	[A,X] ,[B,Y] ,C,Z	Multiply upper vector: $(A*B)_u \rightarrow C$
9B	1	PACKV	A,B,H,O,Z	[A,X] ,[B,Y] ,C,Z	Pack vector: $A_{48\text{-}63}$ & $B_{16\text{-}63} \rightarrow C$ A:exponent, B:coefficient
97	1	RCONV	A,O,Z	[A,X] ,C,Z	Rounded contract vector: $A_{64 \text{ rounded}} \rightarrow C_{32}$
93	1	SQRTV	A,C,H,MA,O,Z	[A,X] ,C,Z	Significant square root vector: $SQRT(A)_s \rightarrow C$
85	1	SUBLV	A,B,MA,MB, N,O,Z	[A,X] ,[B,Y] ,C,Z	Subtract lower vector: $(A - B)_L \rightarrow C$
86	1	SUBNV	A,B,MA,MB, N,O,Z	[A,X] ,[B,Y] ,C,Z	Subtract normalized vector: $(A - B)_n \rightarrow C$
84	1	SUBUV	A,B,MA,MB, N,O,Z	[A,X] ,[B,Y] ,C,Z	Subtract upper vector: $(A - B)_u \rightarrow C$
87	1	SUBXV	A,B,O,Z	[A,X] ,[B,Y] ,C,Z	Subtract index vector: $A_{16\text{-}63} - B_{16\text{-}63} \rightarrow C_{16\text{-}63}$, $A_{0\text{-}15} \rightarrow C_{0\text{-}15}$
90	1	TRUV	A,H,O,Z	[A,X] ,C,Z	Truncate vector: nearest integer .LE. (A) $\rightarrow$ C
98	1	VTOV	A,H,O,Z	[A,X] ,C,Z	Vector to vector transmit: A $\rightarrow$ C

Table C-11. Sparse Vector Instructions

Op	F	Mnemonic	Qualifiers	Operands	Description
A1	2	ADDLS			Add lower sparse vector : $(A + B)_L \rightarrow C$
A2	2	ADDNS			Add normalized sparse vector: $(A + B)_n \rightarrow C$
A0	2	ADDUS			Add upper sparse vector: $(A + B)_u \rightarrow C$
AF	2	DIVSS			Divide significant sparse vector: $(A/B)_s \rightarrow C$
AC	2	DIVUS			Divide upper sparse vector: $(A/B)_u \rightarrow C$
A9	2	MPYLS	C,H,MA, MB,N	$[A_a,X_o]$,$[B_a,Y_o]$,$[C_a,Z_o]$	Multiply lower sparse vector: $(A*B)_L \rightarrow C$
AB	2	MPYSS			Multiply significant sparse vector: $(A*B)_s \rightarrow C$
A8	2	MPYUS			Multiply upper sparse vector: $(A*B)_u \rightarrow C$
A5	2	SUBLS			Subtract lower sparse vector: $(A - B)_L \rightarrow C$
A6	2	SUBNS			Subtract normalized sparse vector: $(A - B)_n \rightarrow C$
A4	2	SUBUS			Subtract upper sparse vector: $(A - B)_u \rightarrow C$

Table C-12. Vector Macro Instructions

Op	F	Mnemonic	Qualifiers	Operands	Description
D1	1	ADJMEAN	H,O,Z	[A,X] ,C,Z	Adjacent mean: $(A_{n+1} + A_n)/2 \rightarrow C_n$
D0	1	AVG	A,B,H,O,Z	[A,X] ,[B,Y] ,C,Z	Vector average: $(A_n + B_n)/2 \rightarrow C_n$
D4	1	AVGD	A,B,H,O,Z	[A,X] ,[B,Y] ,C,Z	Vector average difference: $(A_n - B_n)/2 \rightarrow C_n$
D5	1	DELTA	H,O,Z	[A,X] ,C,Z	Vector delta: $(A_{n+1} - A_n) \rightarrow C_n$
DC	1	DOTV	A,B,H,Z	[A,X] ,[B,Y] ,C_{f-h},Z	Dot product vector: $A \cdot B \rightarrow C, C+1$
DF	1	INTERVAL	H,O,Z	A_{f-h},B_{f-h},C,Z	Interval vector: $A + (n-1)*B \rightarrow C$
DE	1	POLYEVAL	A,H,O,Z	[A,X] ,[B,Y] ,C,Z	Polynomial evaluation: A_n per $B \rightarrow C_n$
DB	1	PRODUCT	H,Z	[A,X] ,C_{f-h},Z	Vector product: $\pi A \rightarrow C$
C0	1	SELEQ			
C2	1	SELGE			Vector select: if A_n .OP. B_n
			A,B,H,Z	[A,X] ,[B,Y] ,C_f,Z	
C3	1	SELLT			Count up to the condition met $\rightarrow$ C
C1	1	SELNE			
DA	1	SUM	H,Z	[A,X] ,C_{f-h},Z	Vector sum: $\Sigma A \rightarrow C, C+1$
B8	1	VREVV	H,O,Z	[A,X] ,C,Z	Transmit vector reversed to vector: $A_{rev} \rightarrow C$
B7	1	VTOVX	B,H	[A,X] ,[B,Y] ,C_a	Transmit vector to vector, destination indexed: $B \rightarrow C$ indexed by A
BA	1	VXTOV	A,H,O,Z	[A,X] ,B_a,C,Z	Transmit vector, source indexed to vector: B indexed by $A \rightarrow C$

Table C-13. String Instructions

Op	F	Mnemonic	Qualifiers	Operands	Description
E0	3	ADDB	none	[A,X] ,[B,Y] ,[C,Z]	Add binary: A + B → C
E4	3	ADDD		[A,X] ,[B,Y] ,[C,Z]	Add decimal: A + B → C
EC	3	ADDMOD		[A,X] ,[C,Y] ,[C,Z] ,I8	Add modulo bytes: $(A_n + B_n)$ mod(I8) → C_n
E8	3	CMPB		[A,X] ,[B,Y]	Compare binary (decimal) set data flags: DFB 53 operands equal; DFB 54 1st operand high; DFB 55 1st operand low
E9	3	CMPD			
E3	3	DIVB		[A,X] ,[B,Y] ,[C,Z]	Divide binary: A/B → C
E7	3	DIVD	none	[A,X] ,[B,Y] ,[C,Z]	Divide decimal: A/B → C
FC	3	DTOZ	NS,SS	[A,X] ,[C,Z]	Unpack BCD to zoned: A → C
EB	3	EMARK	none	[A,X] ,[B,Y] ,[C,Z] ,G	Edit and mark: a per pattern B → C, G = first significant result address
FD	3	MCMPC	D,DD,DM, NIX, NIY	[A,X], [B,Y] ,[C_a,Z]	Compare bytes (character) per mask: find $A_n = B_n$ per mask C, A & B index incremented by number of bytes compared before inequality found
EA	3	MMRGC	none	[A,X] ,[B,Y] ,[C,Z] ,I8	Merge bits per byte (character) mask: A or B per I8 = 0 or 1 → C
F8	3	MOVL	D,DC,DD,DDC, DM,NIX,NIZ	[A,X] ,[C,Z] ,I8	Move bytes left: A → C (left to right); if A short, I8 → C for remaining bytes
F9	3	MOVLC	D,DC,DD,DDC, DM,NIX,NIZ	[A,X] ,[C,Z] ,I8	Move bytes left ones complement: A → C (left to right); if A short, I8 → C for remaining bytes

Table C-13. String Instructions (Cont'd)

Op	F	Mnemonic	Qualifiers	Operands	Description
FA	3	MOVS	none	[A,X],[C,Z],B_f	Move and scale: A → C, scale (B) decimal places
E2	3	MPYB		[A,X],[B,Y],[C,Z]	Multiply binary: A*B → C
E6	3	MPYD		[A,X],[B,Y],[C,Z]	Multiply decimal: A*B → C
D6	3	SRCHKEYB		[A,X],[B,Y],[C,Z],G_f	Search for masked key bits: search A for B per C, A_{index} = # no match
FE	3	SRCHKEYC		[A,X],[B,Y],[C,Z],G_f	Search for masked key chars: search A for B per C, A_{index} = # no match
FF	3	SRCHKEYW		[A,X],[B,Y],[C,Z],G_f	Search for masked key words: search A for B per C, A_{index} = # no match
E1	3	SUBB		[A,X],[B,Y],[C,Z]	Subtract binary: A - B → C
E5	3	SUBD		[A,X],[B,Y],[C,Z]	Subtract decimal: A - B → C
ED	3	SUBMOD	none	[A,X],[B,Y],[C,Z],I8	Modulo subtract bytes: $(A_n - B_n)$ mod(I8) → C_n
EE	3	TL	D,DC,DD,DDC, DM,NIX,NIZ	[A,X],[B,Y],[C,Z]	Translate bytes: B_n → C_n
D7	3	TLMARK	CH,D,DD,DM	[A,X],[B_a,Y],[C,Z]	Translate and mark: A per B → vector C, translate Byte → $C_{exponent}$, partial A field index → $C_{coefficient}$
EF	3	TLTEST	D,DD,DM,NIX	[A,X],[B,Y],Z_f,C_f	Translate and test: B_n → C, A_n → Z if B_n .NE. 0
FB	3	ZTOD	NS,SS	[A,X],[C,Z]	Pack zoned to BCD: A → C

Table C-14. Logical String Instructions

Op	F	Mnemonic	Qualifiers	Operands	Description
F1	3	AND	none	[A,X],[B,Y],[C,Z]	Logical AND: $A \cdot B \rightarrow C$
F6	3	ANDN			Logical AND not: $A \cdot \overline{B} \rightarrow C$
F2	3	IOR			Logical inclusive OR: $A + B \rightarrow C$
F3	3	NAND			Logical NAND: $\overline{A \cdot B} \rightarrow C$
F4	3	NOR			Logical NOR: $\overline{A + B} \rightarrow C$
F5	3	ORN			Logical OR not: $A + \overline{B} \rightarrow C$
F0	3	XOR			Logical exclusive OR: $A - B \rightarrow C$
F7	3	XORN			Logical equivalence (exclusive OR not): $A - \overline{B} \rightarrow C$

Table C-15. Non-Typical Instructions

Op	F	Mnemonic	Qualifiers	Operands	Description
CF	1	ARITHCPS	B,H	[A,X],[B,Y],C_a,Z_o	Arithmetic compress: ABS(A) .GE. $B_n \rightarrow C_n$, set Z_n, O V length $\rightarrow Z_{0\text{-}15}$
04	1	BKPT	none	R_a	Breakpoint: $R_{16\text{-}63} \rightarrow$ breakpoint register
39	A	CLOCK	none	T_f	Transmit (real time clock) $\rightarrow T_{16\text{-}63}$, $0 \rightarrow T_{0\text{-}15}$
C4	1	CMPEQ	A,B,H	[A,X],[B,Y],Z_o	Vector compare, form order vector: if (A_n) .OP. (B_n), set bit Z_n in order vector
C6	1	CMPGE			
C7	1	CMPLT			
C5	1	CMPNE			
1E	7	CNTEQ	none	[R_d,S_i],T_f	Count leading equals: # leading bits equal to bit at [R+S] $\rightarrow T_{48\text{-}63}$
1F	7	CNTO	none	[R_d,S_i],T_f	Count ones in field R: # ones in field [R+S] $\rightarrow T_{48\text{-}63}$
14	7	CPSB	none	R_d,S_L,T_d	Compress bit string: every R_n substring from R_n+S_n pattern $\rightarrow$ T
BC	2	CPSV	H,Z	A_a,C_a,Z_o	Compress vector: vector A $\rightarrow$ sparse C, controlled by O.V. Z
DD	2	DOTS	A,B,H	[A_a,X_o],[B_a,Y_o],$C_{f\text{-}h}$	Sparse vector dot product: A•B $\rightarrow$ C, C+1
06	7	FAULT	none	I5	Simulate fault
1A	7	FILLC	none	I8,[T_d,S_i]	Fill field T with byte (character) R: repeat I8 for field [T+S]
1B	7	FILLR	none	R_f,[T_d,S_i]	Fill field T with byte (R): repeat ($R_{56\text{-}63}$) for field [T+S]

Table C-15. Non-Typical Instructions (Cont'd)

Op	F	Mnemonic	Qualifiers	Operands	Description
* 03	6	KYPT	none	R_a	Keypoint
7E	7	LOD		$[R_a,S_i]\,,T_f$	Load full word: load $[R_a+S_i] \rightarrow T_f$
12	7	LODC		$[R_a,S_i]\,,T_f$	Load byte (character): $[R_a+S_i] \rightarrow T_{56\text{-}63}$, $0 \rightarrow T_{0\text{-}55}$
5E	7	LODH		$[R_a,S_i]\,,T_h$	Load halfword: load $[R_a+S_i] \rightarrow T_h$
16	7	MASKB		R_d,S_d,T_d	Mask bit strings: alternate (R_d) string and (S_d) string $\rightarrow T_{string}$
1D	7	MASKO	none	R_L,S_L,T_d	Form bit mask leading ones: repeat (R_d) ones and (S_d)-(R_d) zeros $\rightarrow T_{string}$
BB	2	MASKV	A,B,H	A_a,B_a,C_a,Z_o	Mask vector: if Z_n=1, $A_n \rightarrow C_n$; if Z_n=0, $B_n \rightarrow C_n$; result length $\rightarrow C_{0\text{-}15}$
1C	7	MASKZ	none	R_L,S_L,T_d	Form mask leading zeros: repeat (R_d) zeros and (S_d)-(R_d) ones $\rightarrow T_{string}$
D8	1	MAX	H,Z	$[A,X]\,,B_f,C_{f\text{-}h},Z$	Vector maximum: $A_{max} \rightarrow$ C, item count $\rightarrow$ B
D9	1	MIN	H,Z	$[A,X]\,,B_f,C_{f\text{-}h},Z$	Vector minimum: $A_{min} \rightarrow$ C, item count $\rightarrow$ B
18	7	MOVR	none	R_i,S_i,T_d	Move bytes right: $(T_d) + (R_i) \rightarrow (T_d) + (R_i) + (S_i)$, bytes moved right $\rightarrow$ left
3D	4	MPYX	none	R_f,S_f,T_f	Multiply index, full word: $R_{16\text{-}63} * S_{16\text{-}63} \rightarrow T_{16\text{-}63}$, $0 \rightarrow T_{0\text{-}15}$
3C	4	MPYXH	none	R_h,S_h,T_h	Multiply index, half-word: $R_{8\text{-}31} * S_{8\text{-}31} \rightarrow T_{8\text{-}31}$, $0 \rightarrow T_{0\text{-}7}$

*Not valid on STAR-100

Table C-15. Non-Typical Instructions (Cont'd)

Op	F	Mnemonic	Qualifiers	Operands	Description
15	7	MRGB	none	R_d,S_d,T_d	Merge bit strings: interleave (R_d) string with (S_d) string → T_d string
17	7	MRGC	none	R_d,S_d,T_d	Merge byte (character) strings: (R_d):(S_d), lesser → T_d
BD	2	MRGV	A,B,H	A_a,B_a,C_a,Z_o	Merge vector: if Z_n=1, A_n → C_n; if Z_n=0, B_n → C_n; result length → $C_{0\text{-}15}$
37	A	RJTIME	none	T_f	Read job interval timer to (T)
28	7	SCANLEQ			Scan left to right from $[T_d,S_i]$ for byte equal to I8, index S_i
29	7	SCANLNE	none	$I8,[T_d,S_i]$	Scan left to right from $[T_d,S_i]$ for byte not equal to I8, index S_i
19	7	SCANRNE			Scan right to left from $[T_d,S_i]$ for byte not equal to I8, decrement S_i
C8	1	SRCHEQ			
CA	1	SRCHGE	H,LH,Z	A,B,C_a,Z	Vector search form indexed list: each (A_n) .OP. (B_n), count → C_n
CB	1	SRCHLT			
C9	1	SRCHNE			
7F	7	STO	none	$[R_a,S_i],T_f$	Store, full word: store (T_f) → address $[R_a+S_i]$
13	7	STOC	none	$[R_a,S_i],T_f$	Store byte (character): $T_{56\text{-}63}$ → address $[R_a+S_i]$
5F	7	STOH	none	$[R_a,S_i],T_h$	Store, half-word: (T_h) → address $[R_a+S_i]$
B9	1	TPMOV	H,O	$[A,X],B_{f\text{-}h},Y_{f\text{-}h},C_a$	Transpose and move 8 by 8 matrix
3A	A	WJTIME	none	R_f	Transmit (R_f) → job interval timer

Table C-16. Monitor Instructions

Op	F	Mnemonic	Qualifiers	Operands	Description
00	4	IDLE	none	none	Idle: enable external interrupts and idle
0D	4	LODAR		none	Load associative registers: full words beginning at $400XX_8 \rightarrow$ AR
0F	4	LODKEY		R_f,S_a,T_a	Load keys from (R_f), translate virtual (S_a) to absolute T_a
0A	4	MTIME		R_f	Transmit $(R_f) \rightarrow$ monitor interval timer
08	4	SETCF		R_f	Input/output: set channel (R_f) channel flag
0C	4	STOAR		none	Store associative registers: AR $\rightarrow 400YY_8$ and higher addresses
0E	4	TLXI		$[R_a,S_i],T_f$	Translate external interrupt: (T_f) = highest priority channel with interrupt, branch to $R_a[S_i]$

Table C-17. Register Designators

Designator	Description
a	a full word register containing an address; length field is ignored
f	full word register containing an operand
h	half word register containing an operand
i	full word register containing an index
d	full word register containing a descriptor
e	full word register whose length field contains an operand
o	full word register containing descriptor of order vector

The 64 bit instructions are assumed:

A, B, C	Descriptors of operands
X, Y	Index
Z	Alone – control vector address in a register pair – index

R, S, T	word in register file
R.J.	right justified
S.E.	sign extended
F.P.	floating point
N/A	not available
none	qualifier not specified
O.V.	order vector
.OP.	arithmetic operator (GE . . . LT . . . LE . . . etc.)

JOB PROCESSING D

This appendix contains a description of the assembler call statement, and the options associated with that statement. Also provided are examples of interactive and batch processing deck set-ups and terminal commands.

ASSEMBLE statement

FORMAT:

```
META  I=SOURCE L=PRINT B=BINARY / 500 I
```

‡fields can be separated by any characters other than 1-9,A-Z or underscore. Blanks can be used as separators

Parameters I, L and B may appear in any order

where

I = source file name – the user must have previously created the source file assigned the name specified. In batch mode the source cards following the control card stream are assumed the input file. The input file may be compressed on expanded.

L = print file name – the print file name is optional and if not specified, listable output will be automatically placed on file "PLIST" by the assembler. When PLIST is used, only the letter L is required, approximately 300 blocks are reserved for PLIST. To print an output listing the user must always specify the following statement: GIVE (output listing file, U = 999999)

B = binary file – this parameter can be omitted if only a syntax check is desired.

EXAMPLE INTERACTIVE ASSEMBLE, LOAD, EXECUTE

1. The assembler deck as shown below was input via the card reader.

```
LOGON 999997 400SDS TESTDECK R S            1
[META SOURCE CARDS
6
 7
  8
   9
```

2. After the assembler deck was read in, at the terminal the following was entered:

```
LOGON 999997 A 400SDS lf*

CREATE(OBJECT02,01,T=P) lf

CREATE(PRINT002,20,T=P) lf
META(I=TESTDECK,L=PRINT002,B=OBJECT2)/ 500 I lf
```

GIVE(PRINT002,U=999999) lf	dispose assembler listing to printer
LOAD / 1000 I lf	Request loader program
INPUT?	Request from loader
OBJECT02	User supplied private file names
ORIGIN?	Request from loader
#28000 lf	First Module loading bit address
ENTRY?	Request from loader
lf	User indicates no options
ANY OTHER OPTIONS?	Request from loader
lf	User indicates LIBRARY option
CONTINUE	Answer from loader
CN = TONY,OU-PRINTMAP lf	User indicates controllee and loadmap option
CONTINUE	Answer from loader
lf	Terminates options and starts load operations
GIVE(PRINTMAP,U=999999)	Dispose loader map to printer
TONY / 500 I	Execute the loaded program
$	

NOTE: the file PRINTMAP is automatically created

* lf = line feed

EXAMPLE BATCH ASSEMBLE, LOAD, EXECUTE

```
(1)             LOGON 999997 400SDS ZBATCH R S B U       ⁚CARD READER ID
(2)  12:00:59   TEST8,T1000.                             ⁚JOB ID
(3)  12:00:59   CREATE(BINARY,02,T=P)                    ⁚FILE CREATION
(4)  12:00:59   CREATE(PLIST,10,T=P)                     ⁚FILE CREATION
(5)  12:00:59   META(I=INPUT,B=BINARY,L=PLIST)           ⁚ASSEMBLE META
(6)  12:01:59   GIVE(PLIST,U=999999)                     ⁚TRANSFER FILE
(7)  12:01:59   LOAD(BINARY,CN=TONY,OU=PMAP)             ⁚LOAD ASSEMBLER OUTPUT
(8)  12:02:59   GIVE(PMAP,U=999999)                      ⁚TRANSFER FILE
(9)  12:02:59   TONY.                                    ⁚EXECUTE CONTROLEE TONY
(10) 12:02:59   $$COMPLETE$$                             ⁚MESSAGE FROM SYSTEM

                7
                 8
                  9
                META DECK
                    .
                    .
                    .
                FINIS
                6
                 7
                  8
                   9
```

(1) The card reader ID card is not field free and variable length names are not allowed.

Columns	Content	Parameter
1-5	LOGON	Card reader ID
7-12	999997	User number
14-19	400SDS	Account number
21-28	ZBATCH	File name
30	R	Record structural file
32	S	Physical file
34	B	Batch processor to be used
36	U	Unrestricted access

(2) Job ID card must contain the job name

TEST 8	Job name
T1000	Time in seconds

(3) Treats a physical file named BINARY

BINARY	File name
10	Length of file in 512 word blocks
T=P	File type in physical data file

(4) The P in PLIST will signal USER1 that file is a print file

PLIST	File name
10	Length of file in 512 word blocks
T=P	File type is physical data file

(5) Assemble META from the card reader and produce binary output on file BINARY and a listing on file PLIST

I=INPUT	Data from unnamed records may be accessed by referencing a file named INPUT in this INPUT in the card reader
B=BINARY	Object code to BINARY
L=PLIST	Assembly listing to PLIST

(6) Transfer the file PLIST to USER1 routine

PLIST	Source file
U=999999	USER1 routine will see that first character of transfer file is P thus a print file

(7) Load the assembler object code into controlee file TONY and place load maps and error messages on PMAP

BINARY	Source file – file to be loaded
CN=TONY	Controlee file is TONY
OU=PMAP	Load maps and error messages on PMAP

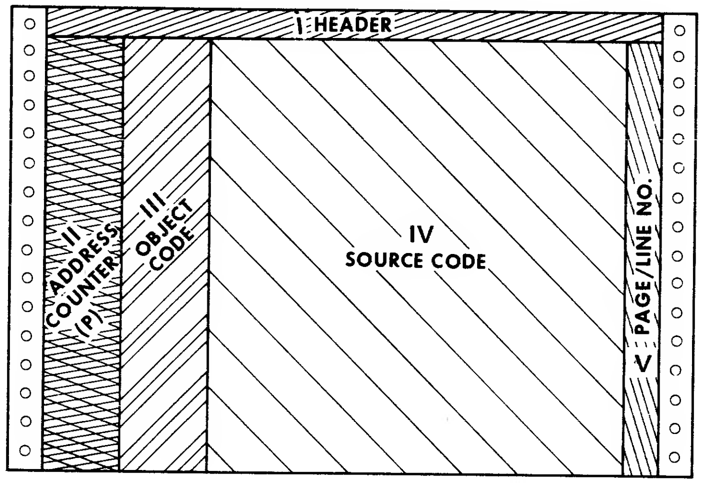
(8) Transfer load maps and error messages to USER1

PMAP	Source file
U=999999	USER1 routine will see that first character of transfer file is P thus a print file

(9) TONY Find this controllee file and execute it

For a more complete description of the control card used in these set-up examples, see the STAR Operating System Reference Manual, Publication No. 60384400.

ASSEMBLY LISTING FORMAT E



I HEADER STAR

FORMAT: ASSEMBLER VER. X.X title PAGE nnnn

The title is blank unless a title is indicated on a TITLE directive.
The nnnn is the page number of the listing.

II ADDRESS (P) COUNTER

FORMAT: RR. VVVVVVVVVVVVV B

RR Hex value of the currently active memory control section ordinal (begins at 01, with a range of 01 to FF)

V's Hex value of the current location counter

B Boundary indicator for current location counter
- F Bit address at FULL word boundary
- H Bit address at HALF word boundary
- C Bit address at BYTE (character) boundary
- B Bit address at BIT boundary
- No effect on location counter

III OBJECT CODE

This field contains the object code in hex.
A maximum of 64 bits of object code appear per line, new lines will be generated for any bits over 64.
Any object code that has relocation will be followed by the ordinal number.
Each field of the object code that has relocation will be on a separate line.

EXAMPLE:

```
B645                  BIM     R_45, label
XXXXXXXXXXXX (n)
      B6   Function code for BIM
      45   Index register 45
      X's  Relocatable address
      n    Ordinal number
```

IV SOURCE CODE

This field contains a copy of the source lines processed.

V PAGE/LINE NO.

This field indicates the page and line number of each source line.

ERROR MESSAGES F

STATEMENT TERMINATING ERROR MESSAGES

UNDEFINED SYMBOL

MULTIPLY DEFINED SYMBOL

ILLEGAL ALIGNMENT VALUE

ILLEGAL OR MISSING LABELS

ILLEGAL OPERAND/PARAMETER

OPERAND NOT A LEGAL SET ELEMENT

MORE THAN 255 EXTERNALS

EXTERNALIZATION NOT ALLOWED AT UNIVERSAL LEVEL

IMPROPER USE OF EXTERNAL OPERAND IN EXPRESSION

FUNCTION NAME USED AS OPERAND

SET NAME USED AS OPERAND IN EXPRESSION

ASSEMBLER'S CAPACITY FOR RELOCATION EXCEEDED

RELOCATABLE TERM ILLEGAL IN EXPRESSION CONTAINING EXTERNAL SYMBOL

IMPROPER USE OF RELOCATABLE TERMS IN EXPRESSION

MULTIPLE RELOCATION ON RESULT OF EXPRESSION

OPERANDS FOR RELATIONAL EXPRESSION HAVE UNLIKE RELOCATION

SUBSCRIBED REFERENCE TO A VARIABLE THAT IS NOT A SET

IMPROPER MODE IN SUBSCRIPT

REPEAT COUNT MISSING/NOT AN INTEGER

IMPROPER NESTING OF REPEATS

IMPROPER MODE ON REPEAT VARIABLE

PROCEDURE LIBRARY I/O ERROR. SEARCH ABORTED.

SYNTAX ERROR IN PROCEDURE/FUNCTION SOURCE STATEMENT, LIBP ABORTED

PROCEDURE/FUNCTION NOT FOUND IN LIBRARY

FILE NAME NOT A 6 CHARACTER SYMBOL

ILLEGAL USE OF .ELM. OPERATOR

IMPROPER USE OF POSITION OPERATOR, (:)

DATA GENERATION ILLEGAL AT UNIVERSAL LEVEL

COMMAND FIELD SYMBOL UNDEFINED AT THIS LEVEL

FORM REFERENCE ILLEGAL AT THIS LEVEL

FUNCTION MAY NOT ALTER P_COUNTER

COMMAND IS NOT A SYMBOL

ILLEGAL NAME FOR PARAMETER SET IN FUNC/PROC STATEMENT

ILLEGAL PASS VALUE

ILLEGAL DATA IN FORM/GEN

MISSING OPERATOR

MODE ERROR IN EXPRESSION

MISSING OPERAND

ILLEGAL SYMBOL

ILLEGAL HEX CONSTANT

ILLEGAL OPERATOR

ILLEGAL STRING CONSTANT

UNMATCHED PAREN

UNMATCHED BRACKET

SYNTAX IS ILLEGAL

OPERAND NOT A CHARACTER STRING CONSTANT

ATTRIBUTE NUMBER OUT OF RANGE

JOB ABORTED, ILLEGAL PARAMETER IN INPUT STATEMENT

EXTRINSIC ATTRIBUTE NOT AN INTEGER VALUE

ILLEGAL TRANSFER ADDRESS IN END STATEMENT

MSEC DOES NOT CORRESPOND, PASS 2 PER PASS 1

DATA DOES NOT CORRESPOND,PASS 2 PER PASS 1

MORE THAN ONE OUTPUT/LISTING STATEMENT IN ASSEMBLY

ILLEGAL PARAMETER IN FUNCTION CALL

REFERENCE TO UNDEFINED ENTRY POINT

SYMBOL NOT A LEGAL OPERAND

TRUNCATED REGISTER VALUE

ILLEGAL VALUE FOR A REGISTER

RELATIVE JUMP OUT OF RANGE

RELATIVE BRANCH TO ADDRESS EXTERNAL TO MSEC

RELOCATABLE OR EXTERNAL DATA DOES NOT END ON WORD BOUNDARY

DATA GENERATED FOR AN EXTERNAL OR RELOCATABLE VALUE LESS THAN 48 BITS

IMPROPER USE OF REAL IN EXPRESSION

ILLEGAL SET STRUCTURE

RELOCATION NOT ALLOWED IN CODE MSEC

OPERATING ON EXTERNALS NOT SUPPORTED BY LOADER

REPEAT SYMBOL REDEFINED IMPROPERLY SYMBOL DROPPED

FORWARD REFERENCE TO REDEFINABLE QUANTITY IS ILLEGAL

ILLEGAL TO REDEFINE DIRECTIVE

WARNING MESSAGES

WARNING – DIVISION BY ZERO INTEGER YIELDS ZERO RESULT, REAL YIELDS INDEFINITE

WARNING – BINARY SCALE FACTOR GREATER THAN 47 APPLIED

WARNING – SUBSCRIPT OUT OF RANGE, NULL ELEMENT USED

WARNING – IDENT/FINIS/ENDP/PROC/FUNC/LIBP CANNOT APPEAR IN REPEAT RANGE

WARNING – TOO MANY ELEMENTS IN LIST, RIGHTMOST ELEMENTS ARE IGNORED

WARNING – LABELS ARE NOT ALLOWED, ANY APPEARING ARE IGNORED

WARNING – GOTO BRANCH NOT PERFORMED, JUMP VALUE NOT AN INTEGER EXPRESSION

WARNING – NO MODIFIERS REQUIRED BY THIS STATEMENT, ANY APPEARING ARE IGNORED

WARNING – ENTRY/EXTERNAL/IDENT CONTAINS MORE THAN 8 CHARACTERS – ONLY FIRST EIGHT RETAINED

WARNING – DEFAULT ASSUMED FOR ILLEGAL MSEC PARAMETER

WARNING – CONSTANT TRUNCATED

WARNING – DATA TRUNCATED

WARNING – REAL EXPONENT OVERFLOW

WARNING – REAL EXPONENT UNDERFLOW

WARNING – POSSIBLE GARBAGE IN FILE

WARNING – TOO MANY PARAMETERS IN FUNCTION CALL, RIGHTMOST PARAMETER IGNORED

WARNING – DATA IMPROPERLY ALIGNED FOR MODE OF OPERAND

WARNING – EXTRA SET ELEMENTS ARE IGNORED

WARNING – MONITOR INSTRUCTION IN JOB MODE MSEC

WARNING – ILLEGAL QUALIFIERS IGNORED

WARNING – DISALLOWED BITS SET IN G FIELD

WARNING – OFFSET/RESULT REGISTER NOT EVEN

WARNING – OVERLAPPING QUALIFIER DEFINITIONS

WARNING – RELATIVE JUMP NOT IN DIRECTION INDICATED

WARNING – FIRST ENTRY IN LABEL FIELD IS AN EXPRESSION

WARNING – BINARY SCALE ON RELOCATABLE ADDRESS

WARNING – POSSIBLE MISSING OPERAND IN INSTRUCTION

WARNING – MISSING QUALIFIER

WARNING – REGISTER VALUE NOT ALIGNED TO APPROPRIATE BOUNDARY

WARNING – AUTOMATIC ALIGNMENT PERFORMED FOR DATA TYPE INDICATED, LABELS MAY NOT CORRESPOND TO START OF DATA

WARNING – LOADER RESTRICTION TRUNCATED TO FIRST EIGHT CHARACTERS

WARNING – DOUBLY DEFINED ENTRY POINT

WARNING – VALUE FROM ANOTHER LEVEL USED FOR . . . .

ASSEMBLER FAILURE MESSAGES

SYSTEM ERROR – S1 – ILLEGAL USE LEVEL IN SYMBOL TABLE

SYSTEM ERROR – S2 – ILLEGAL MODE IN SYMBOL TABLE

SYSTEM ERROR – S3 – ILLEGAL ITEM IN SYMBOL TABLE – DRIVER

SYSTEM ERROR – S4 – LOCATION COUNTER VALUES DO NOT AGREE PASS 2 PER PASS 1

SYSTEM ERROR – S5 – ILLEGAL CHARACTER TRANSLATION VALUE DETECTED – TOKEN

SYSTEM ERROR – S6 – ILLEGAL TOKEN TYPE DETECTED – RPOL

SYSTEM ERROR – S7 – ILLEGAL VALUE FROM COMBINED TOKEN TABLE – TOKEN

SYSTEM ERROR – S8 – MISSING END OR FINIS - - - JOB ABORTED

SYSTEM ERROR – S9 – ILLEGAL TOKEN TYPE DETECTED IN EVAL

SYSTEM ERROR – S10 – ILLEGAL TOKEN NUMBER DETECTED IN EVAL

SYSTEM ERROR – S11 – ILLEGAL SYMBOL TABLE MODE – EVAL

SYSTEM ERROR – S12 – ILLEGAL SYMBOL TABLE ITEM TYPE – EVAL

SYSTEM ERROR – S13 – ZERO LENGTH TOKEN – EVAL

SYSTEM ERROR – S14 – ILLEGAL OPERATOR DETECTED IN RPOL – COMMA

SYSTEM ERROR – S15 – BAD Q ORDINAL ENTRY IN COMMAND TABLE – INST_P

SYSTEM ERROR – S16 – BAD TEMPLATE FOR INSTRUCTION – INST_P

SYSTEM ERROR – S17 – LIMIT FOR EVAL ADDRESS STACK REACHED

SYSTEM ERROR – S18 – LIMIT FOR RPOL OPERAND STACK REACHED

SYSTEM ERROR – S19 – NO SIGN ON ZONED CONSTANT – CONVERSION FUNC

ASSEMBLER PREDEFINED COMMAND-SYMBOLS G

Symbols in the following table have a special meaning to the assembler command field.

Table G-1. Predefined Symbols

Symbol	Function Code or Value (hex)	Use
A	10	Mnemonic qualifier
ABS	79	Instruction mnemonic
ABSH	59	
ABSV	99	
ADDB	E0	
ADDD	E4	
ADDL	61	
ADDLEN	2B	
ADDLH	41	
ADDLS	A1	
ADDLV	81	
ADDMOD	EC	
ADDN	62	
ADDNH	42	
ADDNS	A2	
ADDNV	82	
ADDU	60	
ADDUH	40	
ADDUS	A0	
ADDUV	80	
ADDX	63	
ADDXV	83	
ADJE	75	
ADJEH	55	Instruction mnemonic

Table G-1. Predefined Symbols (continued)

Symbol	Function Code or Value (hex)	Use
ADJEV	95	Instruction mnemonic
ADJMEAN	D1	
ADJS	74	
ADJSH	54	
ADJSV	94	
ALG	05	
AND	F1	
ANDN	F6	
ARITHCPS	CF	Instruction mnemonic
ATT	—	Function name
AVG	D0	Instruction mnemonic
AVGD	D4	Instruction mnemonic
B	08	Mnemonic qualifier
BAB	32	Instruction mnemonic
BADF	33	
BARB	2F	
BEQ	24	
BGE	26	
BHEQ	20	
BHGE	22	
BHLT	23	
BHNE	21	
BIM	B6	
BKPT	04	
BLT	27	
BNE	25	Instruction mnemonic
BR	40	Mnemonic qualifier
BRB	06	Mnemonic qualifier
BRF	04	Mnemonic qualifier
BRIEF	—	Directive
BRO	80	Mnemonic qualifier

Table G-1. Predefined Symbols (continued)

Symbol	Function Code or Value (hex)	Use
BRZ	C0	Mnemonic qualifier
BSAVE	36	Instruction mnemonic
BTOD	11	Instruction mnemonic/ function name
C	02	Mnemonic qualifier
CH	04	Mnemonic qualifier
CLG	72	Instruction mnemonic
CLGH	52	
CLGV	92	
CLOCK	39	
CMPB	E8	
CMPD	E9	
CMPEQ	C4	
CMPGE	C6	
CMPLT	C7	
CMPNE	C5	
CNTEQ	1E	
CNTO	1F	
CON	76	
CONV	96	
CPSB	14	
CPSV	BC	Instruction mnemonic
D	80	Mnemonic qualifier
DBNZ	35	Instruction mnemonic
DC	20	Mnemonic qualifier
DD	C0	Mnemonic qualifier
DDC	30	Mnemonic qualifier
DELTA	D5	Instruction mnemonic
DETAIL	—	Directive
DIVB	E3	Instruction mnemonic

Table G-1. Predefined Symbols (continued)

Symbol	Function Code or Value (hex)	Use
DIVD	E7	Instruction mnemonic
DIVS	6F	
DIVSH	4F	
DIVSS	AF	
DIVSV	8F	
DIVU	6C	
DIVUH	4C	
DIVUS	AC	
DIVUV	8C	Instruction mnemonic
DM	30	Mnemonic qualifier
DOTS	DD	Instruction mnemonic
DOTV	DC	Instruction mnemonic
DTOB	10	Instruction mnemonic
DTOP	—	Function name
DTOZ	FC	Instruction mnemonic
EJECT	—	Directive
ELEN	2A	Instruction mnemonic
EMARK	EB	Instruction mnemonic
END	—	Directive
ENDP	—	
ENTRY	—	
EORG	—	
EQU	—	Directive
ES	3E	Instruction mnemonic
ESH	4D	
EX	BE	
EXH	CD	
EXTB	6E	Instruction mnemonic
EXTC	—	Directive
EXTD	—	Directive
EXIT	09	Instruction mnemonic

Table G-1. Predefined Symbols (continued)

Symbol	Function Code or Value (hex)	Use
EXITP	—	Directive
EXP	7A	Instruction mnemonic
EXPH	5A	
EXPV	9A	
EXTH	5C	
EXTV	9C	
EXTXH	5D	
FAULT	06	
FILLC	1A	
FILLR	1B	Instruction mnemonic
FINIS	—	Directive
FLR	71	Instruction mnemonic
FLRH	51	Instruction mnemonic
FLRV	91	Instruction mnemonic
FORM	—	Directive
FUNC	—	Directive
FF32	—	Function name
F32F	—	Function name
GEN	—	Directive
GOTO	—	Directive
H	80	Mnemonic qualifier
HTOC	—	Function name
IBNZ	31	Instruction mnemonic
IBXEQ	B0	
IBXGE	B2	
IBXGT	B5	
IBXLE	B4	
IBXLT	B3	
IBXNE	B1	Instruction mnemonic
IDENT	—	Directive
IDLE	00	Instruction mnemonic
IMEM	—	Default MSEC name

Table G-1. Predefined Symbols (continued)

Symbol	Function Code or Value (hex)	Use
INPUT	6D	Instruction mnemonic
INSB	—	
INTERVAL	DF	
IOR	F2	
IS	3F	
ISH	4E	Instruction mnemonic
ITOC	—	Function name
ITOF	—	Function name
IX	BF	Instruction mnemonic
IXH	CE	Instruction mnemonic
*KYPT	03	Instruction mnemonic
LIBP	—	Directive
LIST	—	Directive
LISTING	—	Directive
LH	20	Mnemonic qualifier
LOD	7E	Instruction mnemonic
LODAR	0D	
LODC	12	
LODH	5E	
LODKEY	0F	
LSDFR	3B	
LTOL	38	
LTOR	7C	Instruction mnemonic
MA	04	Mnemonic qualifier
MASKB	16	Instruction mnemonic
MASKO	1D	
MASKV	BB	
MASKZ	1C	
MAX	D8	Instruction mnemonic
MB	01	Mnemonic qualifier
MCMPC	FD	Instruction mnemonic
MESSAGE	—	Directive

*Not valid on STAR-100

Table G-1. Predefined Symbols (continued)

Symbol	Function Code or Value (hex)	Use
MIN	D9	Instruction mnemonic
MMRGC	EA	
MOVL	F8	
MOVLC	F9	
MOVR	18	
MOVS	FA	
MPYB	E2	
MPYD	E6	
MPYL	69	
MPYLH	49	
MPYLS	A9	
MPYLV	89	
MPYS	6B	
MPYSH	4B	
MPYSS	AB	
MPYSV	8B	
MPYU	68	
MPYUH	48	
MPYUS	A8	
MPYUV	88	
MPYX	3D	
MPYXH	3C	
MRGB	15	
MRGC	17	
MRGV	BD	Instruction mnemonic
MSEC	—	Directive
MTIME	0A	Instruction mnemonic
N	06	Mnemonic qualifier
NAME	—	Directive
NAND	F3	Instruction mnemonic

Table G-1. Predefined Symbols (continued)

Symbol	Function Code or Value (hex)	Use
NCC	01	Mnemonic qualifier
NIX	04	
NIY	01	
NIZ	01	Mnemonic qualifier
NOLIST	—	Directive
NOR	F4	Instruction mnemonic
NS	C0	Mnemonic qualifier
O	20	Mnemonic qualifier
ORG	—	Directive
ORN	F5	Instruction mnemonic
OUTPUT	—	Directive
PACK	7B	Instruction mnemonic
PACKH	5B	
PACKV	9B	
POLYEVAL	DE	Instruction mnemonic
PROC	—	Directive
PRODUCT	DB	Instruction mnemonic
PTOI	—	Function name
PTOZ	—	Function name
RAND	2D	Instruction mnemonic
RATT	—	Directive
RCON	77	Instruction mnemonic
RCONV	97	Instruction mnemonic
RDEF	—	Directive
RES	—	Directive
RIOR	2E	Instruction mnemonic
RJTIME	37	Instruction mnemonic
RPT	—	Directive
RTOR	78	Instruction mnemonic
RTORH	58	Instruction mnemonic
RXOR	2C	Instruction mnemonic

Table G-1. Predefined Symbols (continued)

Symbol	Function Code or Value (hex)	Use
SCANLEQ	28	Instruction mnemonic
SCANLNE	29	Instruction mnemonic
SCANRNE	19	Instruction mnemonic
SET	—	Directive
SETCF	08	Instruction mnemonic
SELEQ	C0	
SELGE	C2	
SELLT	C3	
SELNE	C1	
SHIFT	34	
SHIFTI	30	Instruction mnemonic
SPACING	—	Directive
SQRT	73	Instruction mnemonic
SQRTH	53	
SQRTV	93	
SRCHEQ	C8	
SRCHGE	CA	
SRCHKEYB	D6	
SRCHKEYC	FE	
SRCHKEYW	FF	
SRCHLT	CB	
SRCHNE	C9	Instruction mnemonic
SS	80	Mnemonic qualifier
STO	7F	Instruction mnemonic
STOAR	0C	
STOC	13	
STOH	5F	
SUBB	E1	
SUBD	E5	
SUBL	65	
SUBHL	45	Instruction mnemonic

Table G-11 Predefined Symbols (continued)

Symbol	Function Code or Value (hex)	Use
SUBLS	A5	Instruction mnemonic
SUBLV	85	
SUBMOD	ED	
SUBN	66	
SUBNH	46	
SUBNS	A6	
SUBNV	86	
SUBU	64	
SUBUH	44	
SUBUS	A4	
SUBUV	84	
SUBX	67	
SUBXV	87	
SUM	DA	
SWAP	7D	Instruction mnemonic
SYM	—	Function name
SZ	30	Mnemonic qualifier
T	10	Mnemonic qualifier
TITLE	—	Directive
TL	EE	Instruction mnemonic
TLMARK	D7	
TLTEST	EF	
TLXI	0E	
TPMOV	B9	
TRU	70	
TRUH	50	
TRUV	90	
VREVV	B8	
VTOV	98	
VTOVX	B7	Instruction mnemonic

Table G-1. Predefined Symbols (continued)

Symbol	Function Code or Value (hex)	Use
VXTOV	BA	Instruction mnemonic
WJTIME	3A	
XOR	F0	
XORN	F7	Instruction mnemonic
XTOD	—	Function name
Z	40	Mnemonic qualifier
ZTOC	—	Function name
ZTOD	FB	Instruction mnemonic
ZTOP	—	Function name

ASSEMBLER LIMITATIONS H

The following limits must be observed:

Maximum symbol length is 63 characters.

Maximum number of memory sections per subprogram is 255.

Maximum number of nested procedures or function calls is 128.

Maximum number of nested subsets is 32.

Maximum number of nested repeat operators is 32.

Maximum number of extrinsic attributes is 120.

Maximum number of nested parentheses in an expression is 60.

EXAMPLES

I

The following examples illustrate a number of the available assembler directives and various machine instruction types. These examples were run on the STAR 65 computer system. A statement of the problem to be solved and a description of the assembler code are provided.

For a description of the register conventions illustrated in the executable examples (vector examples), see appendix E of the STAR Operating System Reference Manual, Publication No. 6038400.

DATA GENERATION

The following examples illustrate three methods of generating data. These examples illustrate the basic use of the following assembler directives.

INPUT	RPT
IDENT	GEN
OUTPUT	END

Example 3 also illustrates the use of functions and sets and is described in detail.

Example 1 – generates integers 1 to 10 at assembly time using the GEN directive.

```
 CDC STAR ASSEMBLER VER 2.2.2                                          DATE: 12SEP74   PAGE   1

                                                  INPUT 1,80                                 1/0001
                                                  IDENT                                      1/0002
01 000000000000 F 00000000 00000001     VALUE     GEN,64 1,2,3,4,5,6,7,8,9,10                1/0003
01 000000000040 F 00000000 00000002
01 000000000080 F 00000000 00000003
01 0000000000C0 F 00000000 00000004
01 000000000100 F 00000000 00000005
01 000000000140 F 00000000 00000006
01 000000000180 F 00000000 00000007
01 0000000001C0 F 00000000 00000008
01 000000000200 F 00000000 00000009
01 000000000240 F 00000000 0000000A
                                                  END                                        1/0004
```

Example 2 – generates integers 1 to 10 at assembly time using the GEN and RPT directives.

```
 CDC STAR ASSEMBLER VER 2.2.2                                          DATE: 12SEP74   PAGE   1

                                               INPUT 1,80                                    1/0001
                                               OUTPUT                                        1/0002
                                               IDENT                                         1/0003
                  DEC            10     A      RPT,10    2                                   1/0004
01 000000000000 F 00000000 00000001     2      GEN    A                                      1/0005
01 000000000040 F 00000000 00000002
01 000000000080 F 00000000 00000003
01 0000000000C0 F 00000000 00000004
01 000000000100 F 00000000 00000005
01 000000000140 F 00000000 00000006
01 000000000180 F 00000000 00000007
01 0000000001C0 F 00000000 00000008
01 000000000200 F 00000000 00000009
01 000000000240 F 00000000 0000000A
                                               END                                           1/0006
```

Example 3 – generates a set of integers 1 to 10 by use of a function.

```
CDC STAR ASSEMBLER VER 2.2.2                                                   DATE: 12SEP74   PAGE    1
                                              INPUT 1,80,1                                             1/0001
                                              OUTPUT                                                   1/0002
                                              TITLE "GENERATE SET VALUES"                              1/0003

CDC STAR ASSEMBLER VER 2.2.2         GENERATE SET VALUES                       DATE: 12SEP74   PAGE    2
                                              FUNC   Z                                                 1/0004
                                     INT      NAME                                                     1/0005
                                     RESULT   SET    1                                                 1/0006
                                     B        RPT,Z(1)-1 1                                             1/0007
                                     1,RESULT SET    .ELM.RESULT,B+1                                   1/0008
                                              ENDP   RESULT                                            1/0009
                                              IDENT                                                    1/0010
01 0000000000 F 00000000 00000001             GEN    .ELM.INT(10)                                      1/0011
01 0000000040 F 00000000 00000002
01 0000000080 F 00000000 00000003
01 00000000C0 F 00000000 00000004
01 0000000100 F 00000000 00000005
01 0000000140 F 00000000 00000006
01 0000000180 F 00000000 00000007
01 00000001C0 F 00000000 00000008
01 0000000200 F 00000000 00000009
01 0000000240 F 00000000 0000000A
                                              END                                                      1/0012
```

In this example, two assembler features are used – functions and sets. The name of the defined function is INT. The function is called in the GEN statement. The call requests the generation of all set elements and passes a value of 10 decimal to list Z in the FUNC statement. Initially, RESULT is set to a value of 1 and then in the RPT statement sets the value of B which is later added to the value 1 in the statement labeled 1, RESULT.

In the RPT statement command list, Z [1]-1 calls for the first element of set Z, which is the value of Z [10] minus 1. This sets the iteration count for the RPT directive. Even though set Z consists of one element, if it were referenced as Z only, a diagnostic would be issued. The final statement in the function definition is the ENDP directive; it specifies that the value assigned to RESULT be returned to the function call statement.

ATTRIBUTE REFERENCING

The ATT directive is illustrated in example 4. The purpose of this example is to determine whether a group of characters constitute a character string. A function is used for character string determination, and the ATT and GOTO statements are illustrated.

```
1  CDC STAR ASSEMBLER VER 2.2.2                                                        DATE: 12SEP74   PAGE   1
0                                      INPUT ,160,20                                                         1/0001
                                       TITLE "CHARACTER_STRING_SIZE_FUNCTION"                                1/0002
1  CDC STAR ASSEMBLER VER 2.2.2    CHARACTER_STRING_SIZE_FUNCTION                      DATE: 12SEP74   PAGE   2
0                                      FUNC  A                                                               1/0003
                                  CHAR_COUNT NAME                                                            1/0004
                                       GOTO,ATT(A[1],2).EQ.7  1                                              1/0005
                                       MESSAGE "NOT A CHARACTER STRING"                                      1/0006
                                       GOTO  2                                                               1/0007
                                  1    GOTO,ATT(A,7).EQ.1   3                                                1/0008
                                       MESSAGE "MORE THAN 1 ELEMENT PASSED"                                  1/0009
                                  2    EXITP 0                                                               1/0010
                                  3    ENDF ATT(A[1],6).ES.-3                                                1/0011
                                       IDENT                                                                 1/0012
 01 0000000000 F 00000000 00000009     GEN   CHAR_COUNT("CHARACTER")                                         1/0013
                                       GEN   CHAR_COUNT(12345)                                               1/0014
 NOT A CHARACTER STRING
 01 0000000040 F 00000000 00000000
                                       GEN   CHAR_COUNT("STAR","ASSEMBLER")                                  1/0015
 MORE THAN 1 ELEMENT PASSED
 01 0000000080 F 00000000 00000000
                                       GEN   CHAR_COUNT("WHAT",,,"FOR")                                      1/0016
 MORE THAN 1 ELEMENT PASSED
 01 00000000C0 F 00000000 00000000
                                       END                                                                   1/0017
1  CDC STAR ASSEMBLER VER 2.2.2    CHARACTER_STRING_SIZE_FUNCTION                      DATE: 12SEP74   PAGE   3
0NUMBER OF WARNING MESSAGES =     0
 NUMBER OF ERROR MESSAGES   =     0
1  CDC STAR ASSEMBLER VER 2.2.2                                                        DATE: 12SEP74   PAGE   4
0                                      FINIS                                                                 1/0018
0
0
0          ASSEMBLY FINISHED
0         3:16 P.M. THURSDAY 12TH. SEPTEMBER, 1974.
0        NUMBER OF STATEMENTS PROCESSED    43
0        NUMBER OF WARNING MESSAGES  NONE
0        NUMBER OF ERROR MESSAGES  NONE
1
```

Example 4. ATT Directive

In the first GOTO statement, (A[1],2) specifies a mode check on the first element of set A. If this element is a character string, the value 7 is returned. (See Intrinsic Attributes in section 5.) If the first element of set A returns a mode value of 7, statement 1 is processed next. Statement 1 also contains an attribute reference ATT (A,7). This reference specifies the 7th attribute of the value assigned to A is to be determined. The 7th attribute requests the number of elements. If > 1, a message is given; if = 1, statement 3 (ENDP) is processed. This statement requests the number of bits assigned to the first element of A shifted right, .BS.-3, 3 places and returned to the call statement (the hexadecimal value 48 assigned to the first element of A (CHARACTER) shifted right results in the value 00000009 across from the GEN statement.

REFERENCING SYMBOLS

Example 5 illustrates the assembly time problem solving capability and the means of referencing a symbol defined with two different values.

```
1  CDC STAR ASSEMBLER VER 2.2.2                                                DATE: 12SEP74    PAGE    1
0                                            INPUT   1,80                                             1/0001
                                             OUTPUT                                                   1/0002
                      00 0000000032   B      RDEF    50                                               1/0003
                                             IDENT                                                    1/0004
                                             FUNC    NUMBER                                           1/0005
                                      SQUARE NAME                                                     1/0006
                                      AGAIN  NAME                                                     1/0007
                                      RESULT RDEF    NUMBER[1]*NUMBER[1]                              1/0008
                                             ENDF    RESULT                                           1/0009
                      00 0000000019   B      RDEF    25                                               1/0010
 01 0000000000 F 0000000000 00000271  C      GEN     SQUARE(B)                                        1/0011
                      00 0000000032   C      RDEF    B$                                               1/0012
 01 0000000040 F 0000000000 000009C4         GEN     AGAIN(C)                                         1/0013
                                             END                                                      1/0014
1  CDC STAR ASSEMBLER VER 2.2.2                                                DATE: 12SEP74    PAGE    2
0NUMBER OF WARNING MESSAGES =       0
 NUMBER OF ERROR MESSAGES   =       0
1  CDC STAR ASSEMBLER VER 2.2.2                                                DATE: 12SEP74    PAGE    3
0                                            FINIS                                                    1/0015
0
0
0            ASSEMBLY FINISHED
0          3:36 P.M. THURSDAY 12TH. SEPTEMBER, 1974.
0         NUMBER OF STATEMENTS PROCESSED     19
0         NUMBER OF WARNING MESSAGES  NONE
0         NUMBER OF ERROR MESSAGES  NONE
1
```

Example 5. Referencing Symbols

The label symbol B is defined with hexadecimal values 32 and 19, and these values are separately passed to function definition (SQUARE). During the first call to SQUARE, the value of B (19) is passed to the function definition set list (NUMBER). The result of the function is returned to the function call level.

Even though there is only one list element in the NUMBER set list, the element must be referenced in the RDEF directive by specifying the element location in [1] brackets. Prior to the second call, the value B is redefined with the value 32. To redefine B, with this value, a $ is appended. The $ instructs the assembler to look for the new value at the Universal level.

CONVERSION FUNCTIONS

ITOC and HTOC conversion function, programmed as part of the assembler, are used in example 6. The ITOC call (line 12) converts an integer string constant (line 7) to a character string constant. The HTOC call (line 15) converts a hexadecimal constant (line 8) to a character string constant. Notice the manner in which the calls are written:

```
                  ┌───────────────────── specifies the set element
      ITOC (I[K])                        to be converted
            │
            └──────────────── specifies the set
```

The HTOC call is written in the same manner.

```
1  CDC STAR ASSEMBLER VER 2.2.2                                                                    DATE: 12SEP74    PAGE    1
0                                                          INPUT 1,80,1                                                  1/0001
                                                           OUTPUT                                                        1/0002
                                                           *TITLE - ASSEMBLER CONVERSION FUNCTIONS                       1/0003
                                                 I         SET  140737488355327,-140737488355327,-256,256,1,-1,0,-0,4096,-4096   1/0004
                                                 H         SET  #FFFFFFFFFFFF,-#FFFFFFFFFFFF,-#F,#F,#0,-#0,#1,-#1,#0123456789,&   1/0005
                                                 -#0123456789,#ABCDEF,-#ABCDEF                                   1/0006
                                                           IDENT                                                         1/0007
                       DEC                10    K          RPT,10  100                                                   1/0008
 01 0000000000 F 31343037 33373438                         GEN     ITOC(I[K])                                            1/0009
 01 0000000040 F 38333535333237
                                                 100       MESSAGE  "-----------------------"                            1/0010
 ------------------------
 01 0000000078 C 20313430 37333734
 01 00000000B8 C 38383335 35333237
 ------------------------
 01 00000000F8 C 20323536
 ------------------------
 01 0000000118 C 323536
 ------------------------
 01 0000000130 C 31
 ------------------------
 01 0000000138 C 2031
 ------------------------
 01 0000000148 C 30
 ------------------------
 01 0000000150 C 30
 ------------------------
 01 0000000158 C 34303936
 ------------------------
 01 0000000178 C 2034303936
 ------------------------
                       DEC                12    H          RPT,12  101                                                   1/0011
 01 00000001A0 H 46464646 46464646                         GEN     HTOC(H[K])                                            1/0012
 01 00000001E0 H 46464646
                                                 101       MESSAGE  "************************"                           1/0013
 ************************
 01 0000000200 F 30303030 30303030
 01 0000000240 F 30303031
 ************************
 01 0000000260 H 46464646 46464646
 01 00000002A0 H 46464631
 ************************
 01 00000002C0 F 30303030 30303030
 01 0000000300 F 30303046
 ************************
 01 0000000320 H 30303030 30303030
 01 0000000360 H 30303030
 ************************
 01 0000000380 F 30303030 30303030
 01 00000003C0 F 30303030
 ************************
 01 00000003E0 H 30303030 30303030
 01 0000000420 H 30303031
 ************************
 01 0000000440 F 46464646 46464646
 01 0000000480 F 46464646
 ************************
1  CDC STAR ASSEMBLER VER 2.2.2                                                                    DATE: 12SEP74    PAGE    2
001 00000004A0 H 30303031 32333435
 01 00000004E0 H 36373839
 ************************
 01 0000000540 F 39383737


 ************************
 01 0000000560 H 30303030 30304142
 01 00000005A0 H 43444546
 ************************
 01 00000005C0 F 46464646 46463534
 01 0000000600 F 30323131
 ************************
                                                           END                                                           1/0014
1  CDC STAR ASSEMBLER VER 2.2.2                                                                    DATE: 12SEP74    PAGE    3
0NUMBER OF WARNING MESSAGES =      0
 NUMBER OF ERROR MESSAGES   =      0
1  CDC STAR ASSEMBLER VER 2.2.2                                                                    DATE: 12SEP74    PAGE    4
0                                                          FINIS                                                         1/0015
0
0
0           ASSEMBLY FINISHED
0          3:24 P.M. THURSDAY 12TH. SEPTEMBER, 1974.
0         NUMBER OF STATEMENTS PROCESSED    54
0         NUMBER OF WARNING MESSAGES  NONE
0         NUMBER OF ERROR MESSAGES  NONE
1
```

Example 6. ITOC Function

SYMBOL CREATION

In example 7, a symbol is generated in the fourth line of the PROC definition. The result generated is R (without quotes) concatenated to the value of N. The 1 following ITOC(N) specifies a $ be appended to the symbol.

The result of the procedure call generates the following:

```
R1          GEN 1
             .
             .
             .
             .
             .
R20         GEN 20
```

```
1  CDC STAR ASSEMBLER VER 2.2.2                                          DATE: 12SEP74   PAGE   1
0                                              OUTPUT                                           1/0001
                                               TITLE "SYMBOL CREATION"                          1/0002
1  CDC STAR ASSEMBLER VER 2.2.2         SYMBOL CREATION                  DATE: 12SEP74   PAGE   2
0                                              IDENT                                            1/0003
                                               PROC    P                                        1/0004
                                       GENRDEF NAME                                             1/0005
                                       N       RPT,P[2]  10                                     1/0006
                                       10,SYM(P[1].CAT.ITOC(N),1) GEN N                         1/0007
                                               ENDP                                             1/0008
 01 0000000000 F 00000000 00000001     CALL    GENRDEF  "R",20                                  1/0009
 01 0000000040 F 00000000 00000002
 01 0000000080 F 00000000 00000003
 01 00000000C0 F 00000000 00000004
 01 0000000100 F 00000000 00000005
 01 0000000140 F 00000000 00000006
 01 0000000180 F 00000000 00000007
 01 00000001C0 F 00000000 00000008
 01 0000000200 F 00000000 00000009
 01 0000000240 F 00000000 0000000A
 01 0000000280 F 00000000 0000000B
 01 00000002C0 F 00000000 0000000C
 01 0000000300 F 00000000 0000000D
 01 0000000340 F 00000000 0000000E
 01 0000000380 F 00000000 0000000F
 01 00000003C0 F 00000000 00000010
 01 0000000400 F 00000000 00000011
 01 0000000440 F 00000000 00000012
 01 0000000480 F 00000000 00000013
 01 00000004C0 F 00000000 00000014
                                               END                                              1/0010
1  CDC STAR ASSEMBLER VER 2.2.2         SYMBOL CREATION                  DATE: 12SEP74   PAGE   3
0NUMBER OF WARNING MESSAGES =     0
 NUMBER OF ERROR MESSAGES   =     0
1  CDC STAR ASSEMBLER VER 2.2.2                                          DATE: 12SEP74   PAGE   4
0                                              FINIS                                            1/0011
0
0
0          ASSEMBLY FINISHED
0        3:26 P.M. THURSDAY 12TH. SEPTEMBER, 1974.
0       NUMBER OF STATEMENTS PROCESSED    33
0       NUMBER OF WARNING MESSAGES  NONE
0       NUMBER OF ERROR MESSAGES  NONE
1
```

Example 7. Symbol Creation

EXECUTABLE EXAMPLES

The following examples include the use of machine instructions, specifically, in the area of vector programming. They are provided to aid in understanding the types of machine instructions available with the STAR computer system. For a description of the register conventions illustrated in these examples, see appendix E of the STAR OS Reference Manual, Publication No. 6038400.

USING VECTORS

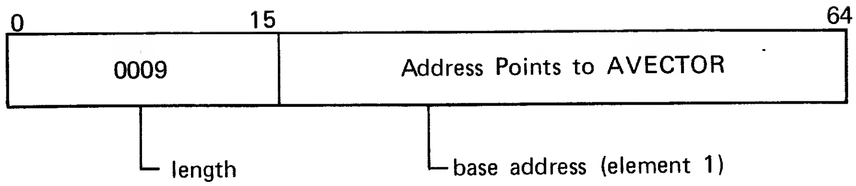
Vector can be created through the GEN directive or by the INTERVAL machine instruction. To create a vector, the programmer must set up a descriptor specifying the length of the vector and the base address (points to the first element of that vector). This descriptor is created in a register the programmer selects in the following order:

base address	An EX instruction for 64-bit register clears 64 bits and enters the base address of the vector specified.
length	An ELEN instruction for 64-bit register enters the length in bits 0-15 of the register.

Example:

```
                INPUT
                OUTPUT
                IDENT
                MSEC          2
A               EQU           #1A*64
                EX            A,AVECTOR
                ELEN          A,9
                  .
                  .
                  .
                MSEC
                  .
                  .
                  .
                  .
AVECTOR         GEN           1,2,3,4,5,6,7,8,9
                  .
                  .
                  .
                END
                FINIS
```

Register #A1



In specifying a register, the user must include the register number times 64 or 32 to specify its size. As described in the STAR Hardware Reference Manual (see Preface), the first half of the register file can be referenced as 128 full-word registers or 256 half-word registers; therefore, full-word register 1E and half-word register 1E are different.

VECTOR ADDITION

The examples which follow illustrate three methods of vector addition:

add index vector

add sparse vector

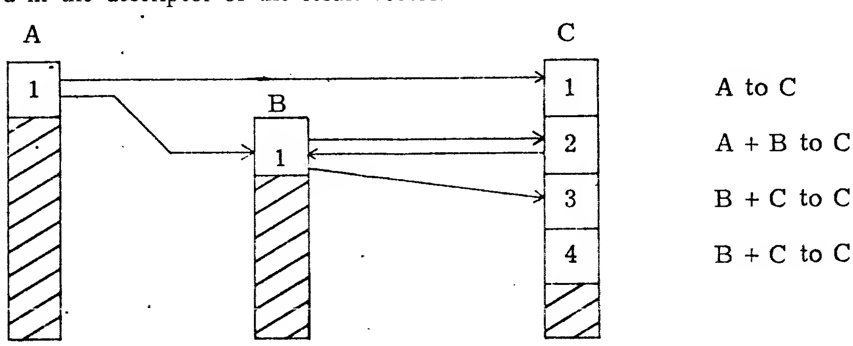
In each example, the vectors are either created differently or the vector descriptors are created with different statement types. The STAR machine is primarily a vector oriented machine, therefore, the use of vectors whenever possible results in savings to the user.

INTERVAL

The INTERVAL statement is a vector macro which executes as follows: The first element created is the value designated in the A source element in the operand field. This value is placed in the C element.

INTERVAL qualifiers (see machine instructions appendix C)	A, B, C, Z

A constant in source operand B is added then to the value of A to form the second element of C. The third to N elements of C are formed by adding the constant in B to preceding element C. The length of the result vector is specified in the descriptor of the result vector.



The qualifiers and Z field, which specifies the constant vector, are not used in the following example and are not discussed here. Control vectors and qualifiers are are illustrated in example 8 which follows.

```
1  CDC STAR ASSEMBLER VER 2.2.2                                                      DATE: 12SEP74   PAGE    1
0                                                    INPUT 1,80,1                                             1/0001
                                                     OUTPUT                                                   1/0002
                                                     TITLE "CREATE VECTORS VIA INTERVAL"                      1/0003
1  CDC STAR ASSEMBLER VER 2.2.2         CREATE VECTORS VIA INTERVAL                  DATE: 12SEP74   PAGE    2
0                                                    IDENT                                                    1/0004
 02 0000000000                                       MSEC    2                                                1/0005
                                                     ENTRY   START                                            1/0006
                               00 0000001000   A     EQU     #40*64   * THESE REGISTERS CONTAIN               1/0007
                               00 0000001040   B     EQU     #41*64   * SOURCE ELEMENTS                       1/0008
                               00 0000001080   C     EQU     #42*64   * CONTAINS RESULT VECTOR DESCRIPTOR     1/0009
                               00 0000000014   N     EQU     20       * LENGTH OF RESULT VECTOR "C"           1/0010
                               00 0000000740   PSP   EQU     #10*64   *                                       1/0011
                               00 0000000540   VITAL EQU     #15*64   *** ENTRY SEQ                           1/0012
                               00 0000000680   RTRN  EQU     #1A*64   *                                       1/0013
 02 0000000000 F 8E420000 05000000   START     EX C,#5000000                                                  1/0014
 02 0000000040 F 8E400000 00000001             EX A,1                                                         1/0015
 02 0000000080 F 78400041                      RTOR A,B   * TRANSMITS VALUE 1 TO B SOURCE                     1/0016
 02 00000000A0 H 2A420014                      ELEN C,N   * VALUE 20 ENTERED INTO LENGTH PORTION OF C DESC.   1/0017
 02 00000000C0 F 0F000040 00410042             INTERVAL A,B,C  *CREATE VECTOR C                               1/0018
 02 0000000100 F 73101500                      SWAP PSP,VITAL                                                 1/0019
 02 0000000120 H 3600001A                      BSAVE   ,RTRN                                                  1/0020
                                               END  START                                                     1/0021
1  CDC STAR ASSEMBLER VER 2.2.2         CREATE VECTORS VIA INTERVAL                  DATE: 12SEP74   PAGE    3
0NUMBER OF WARNING MESSAGES =      0
 NUMBER OF ERROR MESSAGES   =      0
1  CDC STAR ASSEMBLER VER 2.2.2                                                      DATE: 12SEP74   PAGE    4
0                                                    FINIS                                                    1/0022
0
0
0            ASSEMBLY FINISHED
0          3:28 P.M. THURSDAY 12TH. SEPTEMBER, 1974.
0          NUMBER OF STATEMENTS PROCESSED    22
0          NUMBER OF WARNING MESSAGES  NONE
0          NUMBER OF ERROR MESSAGES  NONE
1
```

ADD INTERNAL VECTORS

Example 9 illustrates the use of the INTERNAL macro in generating vectors, the ADDXV instruction, and the use of dynamic space. Also illustrated is the standard entry sequence that should be followed in user programs. Since this subprogram is not called by other routines and does not call any other routine, the entry sequence illustrated is not required. The assignment of the DSP_R register is required, as the results will be entered into the dynamic stack area. Before reading this example, read the Register Conventions in appendix E of STAR OS Reference Manual which provide a description of the register file and the use of the pointers specified in the entry sequence.

In this example, the initial source values are specified by the EX instructions which enter a value of 1 into bits 16-63 of register A and a value of 3 into bits 16-63 of register B1.

Descriptors for the resultant vectors C1, C2, and C3 are then created; length specified is 100 decimal full-words; base address is set at some virtual location in the user available dynamic stack (the locations for vectors C1, C2 and C3 are sequential and 100 full-words apart). Vectors are created by the INTERVAL macro's and then summed by the ADDXZ instruction. For a description of the working of the INTERVAL instruction, see example 8 in this appendix.

```
1   CDC STAR ASSEMBLER VER 2.2.2                                                                DATE: 12SEP74   PAGE    1
0                                                     INPUT 1,80,1                                                      1/0001
                                                      OUTPUT                                                            1/0002
                                                      TITLE "INTERVAL/ ADDXV WITH REGISTER FILE USAGE SEQUENCE"         1/0003
1   CDC STAR ASSEMBLER VER 2.2.2               INTERVAL/ ADDXV WITH REGISTER FILE USAGE SEQUENCE    DATE: 12SEP74   PAGE    2
0                                                     IDENT                                                             1/0004
 02 0000000000                                        MSEC  2                                                           1/0005
                                                      ENTRY START                                                       1/0006
                              00 0000001000   A       EQU   #40*64                                                      1/0007
                              00 0000001040   B1      EQU   #41*64                                                      1/0008
                              00 0000001080   B2      EQU   #42*64                                                      1/0009
                              00 00000010C0   C1      EQU   #43*64                                                      1/0010
                              00 0000001100   C2      EQU   #44*64                                                      1/0011
                              00 0000001140   C3      EQU   #45*64                                                      1/0012
                              00 0000000540   VITAL_R EQU   #15*64  * POINTS TO ENVIRONMENT REGISTERS                   1/0013
                              00 0000000680   RT_N    EQU   #1A*64                                                      1/0014
                              00 00000006C0   DSP_R   EQU   #1B*64  * DYNAMIC SPACE POINTER -POINTS TO NEXT AVAILABLE FREE  1/0015
                              00 0000000700   CSP_R   EQU   #1C*64  * CURRENT STACK POINTER -POINTS TO REG FILE STORAGE     1/0016
                              00 0000000740   PSP_R   EQU   #1D*64  * PREVIOUS STACK POINTER                            1/0017
                                              **** ENTRY SEQUENCE ****                                                  1/0018
 02 0000000000 F                              START                                                                     1/0019
 02 0000000000 F 3E15001A                             ES    VITAL_R,#1A                                                 1/0020
 02 0000000020 H 7000151C                             SWAP  ,VITAL_R,CSP_R                                              1/0021
 02 0000000040 F 781C0010                             RTOR  CSP_R,PSP_R     *CURRENT STACK POINTER EQUALS PREVIOUS      1/0022
 02 0000000060 H 781B001C                             RTOR  DSP_R,CSP_R     *CURRENT STACK POINTER EQUALS DYNAMIC       1/0023
 02 0000000080 F 8F1B0000 00004800                    IX    DSP_R,300*64    *SAVE STACK FRAME SIZE IS 300 WORDS         1/0024
 02 00000000C0 F 8E430000 05000000                    EX    C1,#500000                                                  1/0025
 02 0000000100 F 78430044                             RTOR  C1,C2                                                       1/0026
 02 0000000120 H 3F441900                             IS    C2,100*64       *SET C2 100 FULL WORDS AFTER C1             1/0027
 02 0000000140 F 78440045                             RTOR  C2,C3                                                       1/0028
 02 0000000160 H 3F451900                             IS    C3,100*64       *SET C3 100 FULL WORDS AFTER C2             1/0029
 02 0000000180 F 8E400000 00000001                    EX    A,1                                                         1/0030
 02 00000001C0 F 8E420000 00000003                    EX    B2,3            *SET B3 TO 3                                1/0031
 02 0000000200 F 2A430064                             ELEN  C1,100                                                      1/0032
 02 0000000220 H 2A440064                             ELEN  C2,100                                                      1/0033
 02 0000000240 F 2A450064                             ELEN  C3,100                                                      1/0034
 02 0000000260 H 78400041                             RTOR  A,B1            *PLACE VALUE 1 IN B1                        1/0035
 02 0000000280 F 0F000040 00410043                    INTERVAL A,B1,C1                                                  1/0036
 02 00000002C0 F 0F000040 00420044                    INTERVAL A,B2,C2                                                  1/0037
 02 0000000300 F 93000043 00440045                    ADDXV    C1,C2,C3                                                 1/0038
 02 0000000340 F 7D1D1500                             SWAP     PSP_R,VITAL_R                                            1/0039
 02 0000000360 H 3600001A                             BSAVE    ,RT_N                                                    1/0040
                                                      END      START                                                    1/0041
1   CDC STAR ASSEMBLER VER 2.2.2               INTERVAL/ ADDXV WITH REGISTER FILE USAGE SEQUENCE    DATE: 12SEP74   PAGE    3
0NUMBER OF WARNING MESSAGES =       0
 NUMBER OF ERROR MESSAGES   =       0
1   CDC STAR ASSEMBLER VER 2.2.2                                                                DATE: 12SEP74   PAGE    4
0                                                     FINIS                                                             1/0042
0
0
0        ASSEMBLY FINISHED
0       3:34 P.M. THURSDAY 12TH. SEPTEMBER, 1974.
0       NUMBER OF STATEMENTS PROCESSED    42
0       NUMBER OF WARNING MESSAGES  NONE
0       NUMBER OF ERROR MESSAGES  NONE
1
```

0020000	00000000	00000001	00000000	00000002	00000000	00000003	00000000	00000004
0020100	00000000	00000005	00000000	00000006	00000000	00000007	00000000	00000008
0020200	00000000	00000009	00000000	0000000A	00000000	0000000B	00000000	0000000C
0020300	00000000	0000000D	00000000	0000000E	00000000	0000000F	00000000	00000010
0020400	00000000	00000011	00000000	00000012	00000000	00000013	00000000	00000014
0020500	00000000	00000015	00000000	00000016	00000000	00000017	00000000	00000018
0020600	00000000	00000019	00000000	0000001A	00000000	0000001B	00000000	0000001C
0020700	00000000	0000001D	00000000	0000001E	00000000	0000001F	00000000	00000020
0020800	00000000	00000021	00000000	00000022	00000000	00000023	00000000	00000024
0020900	00000000	00000025	00000000	00000026	00000000	00000027	00000000	00000028
0020A00	00000000	00000029	00000000	0000002A	00000000	0000002B	00000000	0000002C
0020B00	00000000	0000002D	00000000	0000002E	00000000	0000002F	00000000	00000030
0020C00	00000000	00000031	00000000	00000032	00000000	00000033	00000000	00000034
0020D00	00000000	00000035	00000000	00000036	00000000	00000037	00000000	00000038
0020E00	00000000	00000039	00000000	0000003A	00000000	0000003B	00000000	0000003C
0020F00	00000000	0000003D	00000000	0000003E	00000000	0000003F	00000000	00000040
0021000	00000000	00000041	00000000	00000042	00000000	00000043	00000000	00000044
0021100	00000000	00000045	00000000	00000046	00000000	00000047	00000000	00000048
0021200	00000000	00000049	00000000	0000004A	00000000	0000004B	00000000	0000004C
0021300	00000000	0000004D	00000000	0000004E	00000000	0000004F	00000000	00000050
0021400	00000000	00000051	00000000	00000052	00000000	00000053	00000000	00000054
0021500	00000000	00000055	00000000	00000056	00000000	00000057	00000000	00000058
0021600	00000000	00000059	00000000	0000005A	00000000	0000005B	00000000	0000005C
0021700	00000000	0000005D	00000000	0000005E	00000000	0000005F	00000000	00000060
0021800	00000000	00000061	00000000	00000062	00000000	00000063	00000000	00000064
0021900	00000000	00000001	00000000	00000004	00000000	00000007	00000000	0000000A
0021A00	00000000	0000000D	00000000	00000010	00000000	00000013	00000000	00000016
0021B00	00000000	00000019	00000000	0000001C	00000000	0000001F	00000000	00000022
0021C00	00000000	00000025	00000000	00000028	00000000	0000002B	00000000	0000002E
0021D00	00000000	00000031	00000000	00000034	00000000	00000037	00000000	0000003A
0021E00	00000000	0000003D	00000000	00000040	00000000	00000043	00000000	00000046
0021F00	00000000	00000049	00000000	0000004C	00000000	0000004F	00000000	00000052
0022000	00000000	00000055	00000000	00000058	00000000	0000005B	00000000	0000005E
0022100	00000000	00000061	00000000	00000064	00000000	00000067	00000000	0000006A
0022200	00000000	0000006D	00000000	00000070	00000000	00000073	00000000	00000076
0022300	00000000	00000079	00000000	0000007C	00000000	0000007F	00000000	00000082
0022400	00000000	00000085	00000000	00000088	00000000	0000008B	00000000	0000008E
0022500	00000000	00000091	00000000	00000094	00000000	00000097	00000000	0000009A
0022600	00000000	0000009D	00000000	000000A0	00000000	000000A3	00000000	000000A6
0022700	00000000	000000A9	00000000	000000AC	00000000	000000AF	00000000	000000B2
0022800	00000000	000000B5	00000000	000000B8	00000000	000000BB	00000000	000000BE
0022900	00000000	000000C1	00000000	000000C4	00000000	000000C7	00000000	000000CA
0022A00	00000000	000000CD	00000000	000000D0	00000000	000000D3	00000000	000000D6
0022B00	00000000	000000D9	00000000	000000DC	00000000	000000DF	00000000	000000E2
0022C00	00000000	000000E5	00000000	000000E8	00000000	000000EB	00000000	000000EE
0022D00	00000000	000000F1	00000000	000000F4	00000000	000000F7	00000000	000000FA
0022E00	00000000	000000FD	00000000	00000100	00000000	00000103	00000000	00000106
0022F00	00000000	00000109	00000000	0000010C	00000000	0000010F	00000000	00000112
0023000	00000000	00000115	00000000	00000118	00000000	0000011B	00000000	0000011E
0023100	00000000	00000121	00000000	00000124	00000000	00000127	00000000	0000012A
0023200	00000000	00000002	00000000	00000006	00000000	0000000A	00000000	0000000E
0023300	00000000	00000012	00000000	00000016	00000000	0000001A	00000000	0000001E
0023400	00000000	00000022	00000000	00000026	00000000	0000002A	00000000	0000002E
0023500	00000000	00000032	00000000	00000036	00000000	0000003A	00000000	0000003E
0023600	00000000	00000042	00000000	00000046	00000000	0000004A	00000000	0000004E
0023700	00000000	00000052	00000000	00000056	00000000	0000005A	00000000	0000005E
0023800	00000000	00000062	00000000	00000066	00000000	0000006A	00000000	0000006E
0023900	00000000	00000072	00000000	00000076	00000000	0000007A	00000000	0000007E
0023A00	00000000	00000082	00000000	00000086	00000000	0000008A	00000000	0000008E
0023B00	00000000	00000092	00000000	00000096	00000000	0000009A	00000000	0000009E
0023C00	00000000	000000A2	00000000	000000A6	00000000	000000AA	00000000	000000AE

0023D00	00000000	000000B2	00000000	000000B6	00000000	000000BA	00000000	000000BE
0023E00	00000000	000000C2	00000000	000000C6	00000000	000000CA	00000000	000000CE
0023F00	00000000	000000D2	00000000	000000D6	00000000	000000DA	00000000	000000DE
0024000	00000000	000000E2	00000000	000000E6	00000000	000000EA	00000000	000000EE
0024100	00000000	000000F2	00000000	000000F6	00000000	000000FA	00000000	000000FE
0024200	00000000	00000102	00000000	00000106	00000000	0000010A	00000000	0000010E
0024300	00000000	00000112	00000000	00000116	00000000	0000011A	00000000	0000011E
0024400	00000000	00000122	00000000	00000126	00000000	0000012A	00000000	0000012E
0024500	00000000	00000132	00000000	00000136	00000000	0000013A	00000000	0000013E
0024600	00000000	00000142	00000000	00000146	00000000	0000014A	00000000	0000014E
0024700	00000000	00000152	00000000	00000156	00000000	0000015A	00000000	0000015E
0024800	00000000	00000162	00000000	00000166	00000000	0000016A	00000000	0000016E
0024900	00000000	00000172	00000000	00000176	00000000	0000017A	00000000	0000017E
0024A00	00000000	00000182	00000000	00000186	00000000	0000018A	00000000	0000018E

INDEX

Address control 4-27
Address identifier A-16
Arithmetic Operations B-3
ASSEMBLE statement D-1
Assembler failure message F-5
Assembler Limits H-1
Assembly Control 4-7, 4-27
Assembly Listing Format E-1
Assignments (value) 4-20
ATT directive 4-33, 5-3 5-5, I-2
Attribute functions 5-1
Attributes 1-3, 4-33, 5-4, I-2

Batch processing D-3
Binary number representation C-23
Bit string constant A-9
Branch instructions C-9, C-31
BRIEF directive 4-6, 4-51
Broadcast element C-10, C-13

Character set A-2, A-3
Character string constnat A-10
Code section 2-1, 2-3
Coding conventions 2-2
Command field 3-1, C-1
Comment field 3-1
Commercial at @ 4-24, 4-27
Common section (subprogram) 2-1, 2-3
Conditional assembly 1-1, 4-7
Constants A-4, A-14
Continuation (statement) 3-1
Control vectors C-10, C-13, I-9
Conversion functions 5-1, I-4

Data generation 4-23, 4-42, I-1
Data Section (subprograms) 2-1, 2-3
Default MSEC (IMEM) 4-29
DETAIL directive 4-6, 4-51
Delimiter characters A-3, C-24
Directive names A-16, A-18

EJECT directive 4-4, 4-51
Element and sub-element referencing 4-19
ENDP directive 4-16, 4-37, 4-46, 4-54, I-2
END directive 4-7, 4-52, I-1,
ENTRY directive 4-13, 4-52
EORG directive 4-32, 4-53
EQU directive 4-21, 4-47, 4-53
Error Messages F-1
Evaluation of expressions B-3
Examples I-1
Executive Output 1-5
EXITP directive 4-16, 4-44, 4-46, 4-54
Expressions B-1
Expression and Mode Evaluation B-4
Externals (EXTC, EXTD) 4-14, 4-24, 4-52
Extrinsic Attributes (RATT) 4-33

FINIS directive 2-1, 4-7, 4-52
FORM directive 2-2, 4-23, 4-53
FORM names A-16, A-18
Form Referencing 4-24, 4-53
Function names A-16, A-17
Function references 4-46, 4-54
Functions 1-2, 4-44
Function definition (conventions) 4-44
FUNC directive 4-5, 4-47, 4-54, I-2

GEN directive 4-25, 4-47, 4-53, I-1
GOTO directive 4-10, 4-52

Hardware or Assembler errors 1-6, F-5
Hexadecimal constant A-7
Hexadecimal string constnat A-8
Hierarchical expression evaluation B-9

IDENT directive 2-1, 4-6, 4-47, 4-52, I-1
Index incrementing C-24
Index instructions C-9, C-27

In-line PROC	4-34
INPUT directive	3-1, 4-2, 4-51, I-1
Instruction designator	C-6
Instruction mnemonic	A-19
Integer constant	A-5
Integer string constant	A-6
Interactive processing	D-1
Interval instruction	I-9
Interval vector statement	I-8, I-9
Intrinsic attributes (ATT)	5-4, I-3
Label field	3-1, C-1
Level	2-4, 2-5
LIBP directive	4-3, 4-51
Limitations (assembler)	H-1
Listing control	4-4
LIST directive	4-5, 4-51
LISTING directive	4-3, 4-51
Location control	1-3, 4-27
Location-independent code	2-2
Logical operations	B-8
Logic string instructions	C-25, C-39
Machine instructions	4-27, C-1, I-7
Machine instruction designators	C-6 thru C-8
Machine instruction formats	C-1 thru C-5
Machine instruction types	C-8
MESSAGE directive	4-5, 4-51
MSEC Default	4-28
MSEC directive	2-2, 4-29, 4-53
Monitor instructions	C-25, C-43
NAME directive	4-36, 4-45
Nested procedures	4-29
NOLIST directive	4-5, 4-51
Non-typical instructions	C-17, C-40
NOPH procedure	5-1, 5-7
Null elements	4-18
Numeric label	A-19
Offset number	C-11 thru C-13
Operand field	3-1, C-3
Operators	B-2
Order vector	C-10
ORG directive	4-31, 4-53
OUTPUT directive	4-2, 4-51, I-1
Packed decimal constant	A-10
Packed decimal data strings	C-23
Positional operator	4-17
Predefined command symbols	G-1
Printer Output	1-5
PROC directive	4-36, 4-54
Procedure	2-2, 4-34
Procedure definition	4-36
Procedure name	4-35, A-16, A-18
Procedure reference	4-27, 4-38, 4-43, 4-54
Procedure reference termination	4-38
Program conventions	2-2
Qualifiers	C-1
RATT directive	4-33, 4-54
RDEF directive	2-4, 4-20, 4-25, 4-48, 4-53, 5-3
Real constant	A-12
Redefining a symbol	I-4
Re-entrant code	2-2
Referencing attributes	4-34
Referencing sets	4-19
Referencing elements and subelements	4-19
Referencing Forms	4-24
Register Designators	C-44
Register instructions	C-9, C-28
Relational operations	B-3, B-7
Repetition factor	4-18
RES directive	4-31, 4-53
RPT directive	4-8, 4-12, 4-52, I-1
SET directive	4-16, 4-25, 4-48, 4-52
Sets	1-2
Set name	A-16, A-18
SHORTBR procedure	5-1, 5-7
Sign control	C-11, C-25
Significance count	C-11
Source statement errors	1-5
SPACING directive	4-4, 4-51
Sparse vector	C-10, C-14, C-17 thru C-22, C-35
Special characters	A-4
Standard input	1-4

STAR machine instructions	C-26 thru C-43
Statement boundaries	3-1
Statement format	3-2
Statement terminating errors	1-5, F-1
String instructions	C-24, C-37 C-38
String instruction delimiters	C-24
Sub-element reference	4-19
Subprogram linking	4-13
Subprogram area	2-1, 2-3, 4-29
SYM function	5-3
Symbols	1-3, A-16
Symbol creation function	5-1, I-6
Symbol definition	2-4, 4-44, I-4, I-6
Symbol redefinition	I-4
Symbol reference	4-35, 4-44, I-4
Symbol reference levels	2-4, 4-44
TITLE directive	4-5, 4-51
Universal area	2-1, 4-35
Value assignment	4-20 thru 4-22
Variable identifier	A-16
Vector addition	C-14, C-17, C-22, I-9
Vector creation	I-9
Vector generation	I-7
Vector instructions	C-10, C-15, C-33
Vector macros	C-13
Vector macro instructions	C-36
Vector matrix	C-14
Vector offsets	C-12
Warning messages	1-6, F-3
Zoned decimal constants	A-11
Zoned decimal data strings	C-23

CUT ON THIS LINE

COMMENT SHEET

CONTROL DATA CORPORATION

TITLE: STAR Computer System, STAR Assembler Reference Manual

PUBLICATION NO. 19980200 REVISION B

This form is not intended to be used as an order blank. Control Data Corporation solicits your comments about this manual with a view to improving its usefulness in later editions.

Applications for which you use this manual.

Do you find it adequate for your purpose?

What improvements to this manual do you recommend to better serve your purpose?

Note specific errors discovered (please include page number reference).

General comments:

FROM NAME: ______________________ POSITION: ______________________

COMPANY NAME: ______________________

ADDRESS: ______________________

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

STAPLE

STAPLE

FOLD

FOLD

FIRST CLASS
PERMIT NO. 8241

MINNEAPOLIS, MINN.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION
Documentation Department
215 Moffett Park Drive
Sunnyvale, California 94086

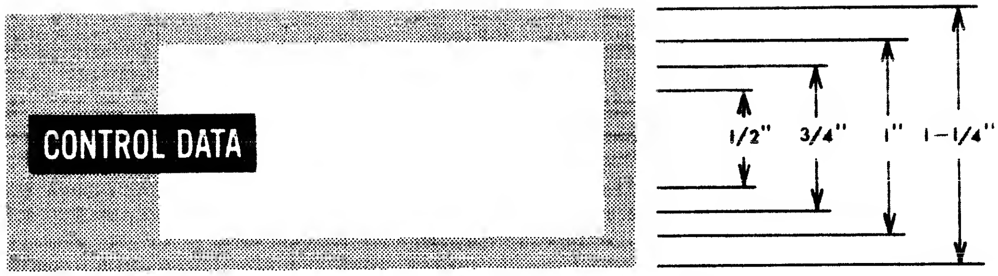
CUT ON THIS LINE

FOLD

FOLD

STAPLE

STAPLE



→→ CUT OUT FOR USE AS LOOSE-LEAF BINDER TITLE TAB

CONTROL DATA
CORPORATION

CORPORATE HEADQUARTERS, 8100 34th AVE. SO., MINNEAPOLIS, MINN. 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

PRINTED IN U.S.A.